Project 1

主要是定位到EP0、EP1、CH相关汇编，以及reverse部分。

环境配置

- 宿主机操作系统：Windows 10
- 虚拟机软件：VMware Workstation 16 Pro
- 虚拟机操作系统：Ubuntu 18.04.6 LTS

https://github.com/riscv-collab/riscv-gnu-toolchain

```
$ git clone https://github.com/riscv/riscv-gnu-toolchain --recursive
#这里不加recursive的话，gcc等都会在make时下载，但无法看到下载的进度，所以还是直接把所有依赖都下载下来比较好

$ sudo apt-get install autoconf automake autotools-dev curl python3 libmpc-dev libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool patchutils bc zlib1g-dev libexpat-dev

$ export PATH=$PATH:/opt/riscv/bin
$ ./configure --prefix=/opt/riscv --with-arch=rv32gc --with-abi=ilp32d
$ sudo make -j16
#必须sudo，否则写文件时，有些文件夹无权访问，最好加上线程，否则会非常慢
#此处如果用make linux的话，编译sha256.s生成的文件将报错cannot execute binary file: Exec format error
```

任务1：仿真器拓展

1. instforms: 指令编码

```
// instforms.cpp
// 根据指令设计，对照encodeCube函数，只需要更改funct3的值即可
// 同时需要在对应头文件instforms.hpp中添加函数声明，此步已完成，不需要自己更改
bool
RFormInst::encodeCube(unsigned rdv, unsigned rs1v, unsigned rs2v)
{
  if (rdv > 31 or rs1v > 31 or rs2v > 31)
    return false;
  bits.opcode = 0x33;
  bits.rd = rdv & 0x1f;
  bits.funct3 = 0;
  bits.rs1 = rs1v & 0x1f;
  bits.rs2 = rs2v & 0x1f;
  bits.funct7 = 2;
  return true;
}

bool
RFormInst::encodeRotleft(unsigned rdv, unsigned rs1v, unsigned rs2v)
{
```

```
  if (rdv > 31 or rs1v > 31 or rs2v > 31)
    return false;
  bits.opcode = 0x33;
  bits.rd = rdv & 0x1f;
  bits.funct3 = 1;
  bits.rs1 = rs1v & 0x1f;
  bits.rs2 = rs2v & 0x1f;
  bits.funct7 = 2;
  return true;
}

bool
RFormInst::encodeRotright(unsigned rdv, unsigned rs1v, unsigned rs2v)
{
  if (rdv > 31 or rs1v > 31 or rs2v > 31)
    return false;
  bits.opcode = 0x33;
  bits.rd = rdv & 0x1f;
  bits.funct3 = 2;
  bits.rs1 = rs1v & 0x1f;
  bits.rs2 = rs2v & 0x1f;
  bits.funct7 = 2;
  return true;
}

bool
RFormInst::encodeReverse(unsigned rdv, unsigned rs1v, unsigned rs2v)
{
  if (rdv > 31 or rs1v > 31 or rs2v > 31)
    return false;
  bits.opcode = 0x33;
  bits.rd = rdv & 0x1f;
  bits.funct3 = 3;
  bits.rs1 = rs1v & 0x1f;
  bits.rs2 = rs2v & 0x1f;
  bits.funct7 = 2;
  return true;
}

bool
RFormInst::encodeNotand(unsigned rdv, unsigned rs1v, unsigned rs2v)
{
  if (rdv > 31 or rs1v > 31 or rs2v > 31)
    return false;
  bits.opcode = 0x33;
  bits.rd = rdv & 0x1f;
  bits.funct3 = 4;
  bits.rs1 = rs1v & 0x1f;
  bits.rs2 = rs2v & 0x1f;
  bits.funct7 = 2;
  return true;
}
```

2. decode: 指令译码

```
// InstEntry.cpp
// 因为都是R型指令，所以只需要改变name，id和code，其余和cube保持一致即可
      { "cube", InstId::cube, 0x4000033, top7Funct3Low7Mask,
    InstType::Int,
    OperandType::IntReg, OperandMode::Write, rdMask,
    OperandType::IntReg, OperandMode::Read, rs1Mask,
    OperandType::IntReg, OperandMode::Read, rs2Mask },

      { "rotleft", InstId::rotleft, 0x4001033, top7Funct3Low7Mask,
    InstType::Int,
    OperandType::IntReg, OperandMode::Write, rdMask,
    OperandType::IntReg, OperandMode::Read, rs1Mask,
    OperandType::IntReg, OperandMode::Read, rs2Mask },

      { "rotright", InstId::rotright, 0x4002033, top7Funct3Low7Mask,
    InstType::Int,
    OperandType::IntReg, OperandMode::Write, rdMask,
    OperandType::IntReg, OperandMode::Read, rs1Mask,
    OperandType::IntReg, OperandMode::Read, rs2Mask },

      { "reverse", InstId::reverse, 0x4003033, top7Funct3Low7Mask,
    InstType::Int,
    OperandType::IntReg, OperandMode::Write, rdMask,
    OperandType::IntReg, OperandMode::Read, rs1Mask,
    OperandType::IntReg, OperandMode::Read, rs2Mask },

      { "notand", InstId::notand, 0x4004033, top7Funct3Low7Mask,
    InstType::Int,
    OperandType::IntReg, OperandMode::Write, rdMask,
    OperandType::IntReg, OperandMode::Read, rs1Mask,
    OperandType::IntReg, OperandMode::Read, rs2Mask },

// InstId.hpp
// 在枚举中增加对应指令，并且修改maxId为notand
     cube,
     rotleft,
     rotright,
     reverse,
     notand,

     maxId = notand,


// decode.cpp
// decode.cpp中，通过funct3来区分我们设计的不同指令
  else if(funct7 == 2)
    {
      // cube
      if (funct3 == 0) return instTable_.getEntry(InstId::cube);
      // rotleft
      if (funct3 == 1) return instTable_.getEntry(InstId::rotleft);
      // rotright
      if (funct3 == 2) return instTable_.getEntry(InstId::rotright);
```

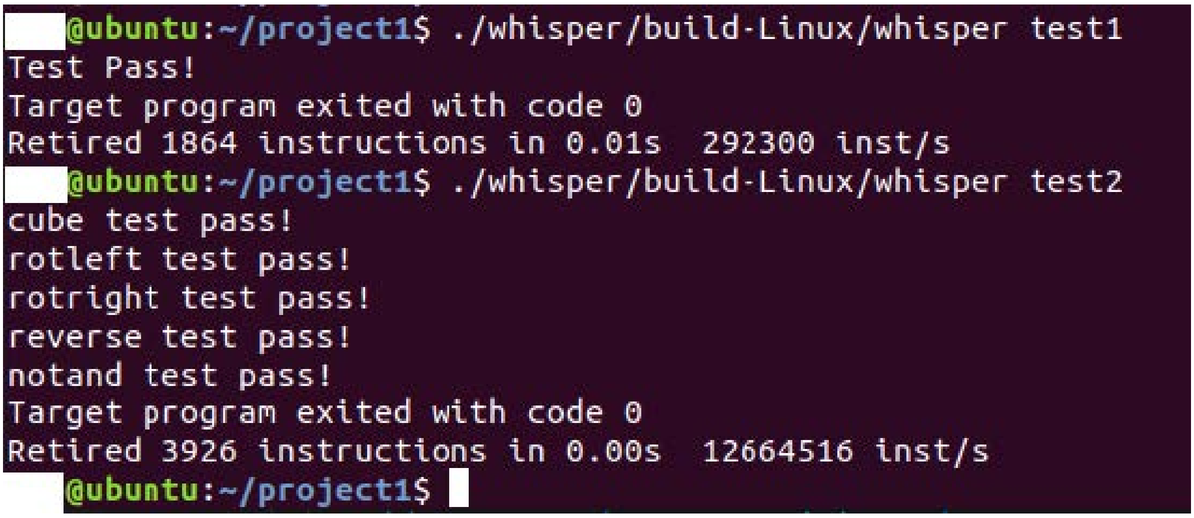
```
        // reverse
        if (funct3 == 3) return instTable_.getEntry(InstId::reverse);
        // notand
        if (funct3 == 4) return instTable_.getEntry(InstId::notand);
      }
```

3. Hart: 指令执行

```
// Hart.hpp
// 在Hart.hpp中添加对应函数的声明
    void execCube(const DecodedInst*);
    void execRotleft(const DecodedInst*);
    void execRotright(const DecodedInst*);
    void execReverse(const DecodedInst*);
    void execNotand(const DecodedInst*);

// Hart.cpp
// 用intRegs_.read来读取值，intRegs_.write来写值
// 要注意左移和右移需要转换成32位无符号整数，否则会出错
// 此外，在Hart.cpp中要记得添加拓展指令的 label 以及相应的跳转执行，目前不需要，但是拓展指令时需要
template <typename URV>
inline
void
Hart<URV>::execCube(const DecodedInst* di)
{
  URV v = intRegs_.read(di->op1()) * intRegs_.read(di->op1()) *
intRegs_.read(di->op1());
  intRegs_.write(di->op0(), v);
}


template <typename URV>
inline
void
Hart<URV>::execRotleft(const DecodedInst* di)
{
  uint32_t rs1 = intRegs_.read(di->op1());
  uint32_t rs2 = intRegs_.read(di->op2());
  URV v = (rs1 << rs2) | (rs1 >> (32 - rs2));
  intRegs_.write(di->op0(), v);
}

template <typename URV>
inline
void
Hart<URV>::execRotright(const DecodedInst* di)
{
  uint32_t rs1 = intRegs_.read(di->op1());
  uint32_t rs2 = intRegs_.read(di->op2());
  URV v = (rs1 >> rs2) | (rs1 << (32 - rs2));
  intRegs_.write(di->op0(), v);
}

template <typename URV>
```

```
inline
void
Hart<URV>::execReverse(const DecodedInst* di)
{
  URV rs1 = intRegs_.read(di->op1());
  URV rs2 = intRegs_.read(di->op2());
  URV v = (rs1 >> (24 - rs2*8)) & 0x000000ff;
  intRegs_.write(di->op0(), v);
}

template <typename URV>
inline
void
Hart<URV>::execNotand(const DecodedInst* di)
{
  URV rs1 = intRegs_.read(di->op1());
  URV rs2 = intRegs_.read(di->op2());
  URV v = ~(rs1)&rs2;
  intRegs_.write(di->op0(), v);
}
```

4. 测试



任务2：优化哈希加密

1.优化

```
#   类型一 用我们设计的循环左移来替代原本的操作。原操作是分别左移右移最后对结果做或运算
#   但是使用我们设计的指令，需要把左移的位数先存到一个寄存器中，作为rs2传入
#   所以，原本需要三条汇编减少为两条，但是因为循环，以及宏定义EP0，EP1中都有涉及，所以是指令减少的最关键的部分，对应C语言sha256_transform函数中的m[i] = (data[j] << 24) | (data[j + 1] << 16) | (data[j + 2] << 8) | (data[j + 3]); t1 = h + EP1(e) + CH(e,f,g) + k[i] + m[i]; t2 = EP0(a) + MAJ(a,b,c);
#   具体为
#   slli    a3,a5,13
#   srli    a4,a5,19
#   or  a4,a4,a3
    addi a3,x0,13
    .insn r 0x33,1,2,a4,a5,a3
```

```
#   类型二 notand
#   除lw读值的指令，我们可以把原本拆分为两部的与非操作，用我们的notand指令来替代
#   not a2,a5
#   and a5,a2,a5
#   .insn r 0x33,4,2,a5,a2,a5

#   类型三 reverse
#   C语言中sha256_final最后hash[i]=(ctx->state[0] >> (24 - i * 8)) & 0x000000ff;反转字节来输出。
#   这部分反转字节的代码对应汇编中的.L21
#   sub a5,a0,a1
#   slli    a5,a5,3
#   srl a4,a4,a5    这三句可以等效替代为以下的汇编
.insn r 0x33,3,2,a4,a4,a1

#   类型四 将常量分寄存器保存，不需要重复读取
```

```
#   this is the reverse part
#   lw  a5,-68(s0)
    lw  a2,-68(s0)
#   lw  a4,80(a5)    use a2 to save the constant offset
    lw  a4,80(a2)
#   li  a3,3         use a0 as constant 3
    li  a0,3
#   lw  a5,-52(s0)   use a1 to save the constant value -52(s0)
    lw  a1,-52(s0)
##  sub a5,a0,a1
##  slli    a5,a5,3
##  srl a4,a4,a5
#   lw  a4,-72(s0)   use a3 to save the constant value -72(s0)
    lw  a3,-72(s0)
##  addi    a5,a5,0
#   lw  a5,-52(s0)
    .insn r 0x33,3,2,a4,a4,a1
    add a5,a3,a1
##  andi    a4,a4,0xff
    sb  a4,0(a5)
#   lw  a5,-68(s0)
    lw  a4,84(a2)
#   li  a3,3
#   lw  a5,-52(s0)
##  sub a5,a0,a1
##  slli    a5,a5,3
##  srl a4,a4,a5
    .insn r 0x33,3,2,a4,a4,a1
```

```
#   以reverse部分汇编为例，我们用a1,a2,a3,a5来分别保存3,-52(s0),-72(s0),-68(s0)这些常量值，这样就只需要读取一次
#   因为循环的存在，可以节省大量的汇编代码
```

2.验证

```
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256
Please input string: Zhejiang University
hash hex: 6e908a6810a2609c1e9d5653f1aed04de19a187a520ed38968823bd3b12ee511
Target program exited with code 0
Retired 30399 instructions in 20.66s  1471 inst/s
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256opt
Please input string: Zhejiang University
hash hex: 6e908a6810a2609c1e9d5653f1aed04de19a187a520ed38968823bd3b12ee511
Target program exited with code 0
Retired 27802 instructions in 5.40s  5151 inst/s
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256
Please input string: COD
hash hex: ca8c6373fbefb853775c35e5c35ee8020b6912b07b5728333b0b4134ab9cc373
Target program exited with code 0
Retired 30332 instructions in 2.14s  14168 inst/s
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256opt
Please input string: COD
hash hex: ca8c6373fbefb853775c35e5c35ee8020b6912b07b5728333b0b4134ab9cc373
Target program exited with code 0
Retired 27745 instructions in 1.85s  14993 inst/s
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256
Please input string: [redacted]
hash hex: f8116d7ad87ec81dba646a649be1754d2a02c0b86bcafa877ae98c9bb0788ba3
Target program exited with code 0
Retired 30474 instructions in 9.32s  3268 inst/s
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256opt
Please input string: [redacted]
hash hex: f8116d7ad87ec81dba646a649be1754d2a02c0b86bcafa877ae98c9bb0788ba3
Target program exited with code 0
Retired 27879 instructions in 4.58s  6083 inst/s
[redacted]@ubuntu:~/project1/sha256$
```

	Zhejiang University	COD	学号	avg
sha256	30399	30332	30474	30401.67
sha256opt	27802	27745	27879	27808.67

结果正确，平均减少2593条指令。

3.拓展指令

```
// 再主循环中找了两个相邻指令捏合成一条，预期可以减少64条指令
// step 1: instforms.hpp中增加解码函数声明 bool encodeXoradd(unsigned rd, unsigned rs1, unsigned rs2);
// step 2: instforms.cpp中写对应函数实现
// step 3: InstEntry.cpp中增加对应条目
// step 4: InstId.hpp增加对应指令label的枚举量，并修改maxId
// step 5: decode.cpp中增加对应译码内容，即通过funct7等值对指令进行区别
// step 6: Hart.hpp中增加对应执行函数声明
// step 7: Hart.cpp中完成实际执行运算函数的定义
// step 8: Hart.cpp添加拓展指令的 label 以及相应的跳转执行(有两处！)
```

```
#   xor a1,a3,a1
#   add a1,a4,a1
    .insn r 0x33,5,2,a1,a3,a4
#   将异或和加和用R指令合并成一个，对应执行函数如下：
    Hart<URV>::execXoradd(const DecodedInst* di)
    {
      uint32_t rd = intRegs_.read(di->op0());
      uint32_t rs1 = intRegs_.read(di->op1());
      uint32_t rs2 = intRegs_.read(di->op2());
      uint32_t v = (rd ^ rs1) + rs2;
      intRegs_.write(di->op0(), v);
    }
```

4.验证

```
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256
Please input string: Zhejiang University
hash hex: 6e908a6810a2609c1e9d5653f1aed04de19a187a520ed38968823bd3b12ee511
Target program exited with code 0
Retired 30399 instructions in 5.36s  5669 inst/s
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256opt
Please input string: Zhejiang University
hash hex: 6e908a6810a2609c1e9d5653f1aed04de19a187a520ed38968823bd3b12ee511
Target program exited with code 0
Retired 27738 instructions in 5.01s  5538 inst/s
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256
Please input string: COD
hash hex: ca8c6373fbefb853775c35e5c35ee8020b6912b07b5728333b0b4134ab9cc373
Target program exited with code 0
Retired 30332 instructions in 1.89s  16049 inst/s
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256opt
Please input string: COD
hash hex: ca8c6373fbefb853775c35e5c35ee8020b6912b07b5728333b0b4134ab9cc373
Target program exited with code 0
Retired 27681 instructions in 2.05s  13505 inst/s
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256
Please input string: [redacted]
hash hex: f8116d7ad87ec81dba646a649be1754d2a02c0b86bcafa877ae98c9bb0788ba3
Target program exited with code 0
Retired 30474 instructions in 2.63s  11590 inst/s
[redacted]@ubuntu:~/project1/sha256$ ../whisper/build-Linux/whisper sha256opt
Please input string: [redacted]
hash hex: f8116d7ad87ec81dba646a649be1754d2a02c0b86bcafa877ae98c9bb0788ba3
Target program exited with code 0
Retired 27815 instructions in 3.00s  9281 inst/s
[redacted]@ubuntu:~/project1/sha256$
```

	Zhejiang University	COD	学号	avg
sha256	30399	30332	30474	30401.67
sha256opt	27738	27681	27815	27744.67

结果正确，平均减少2657条指令。