从零开始的RISC-V模拟器开发

第2讲 Spike篇之CPU模拟2

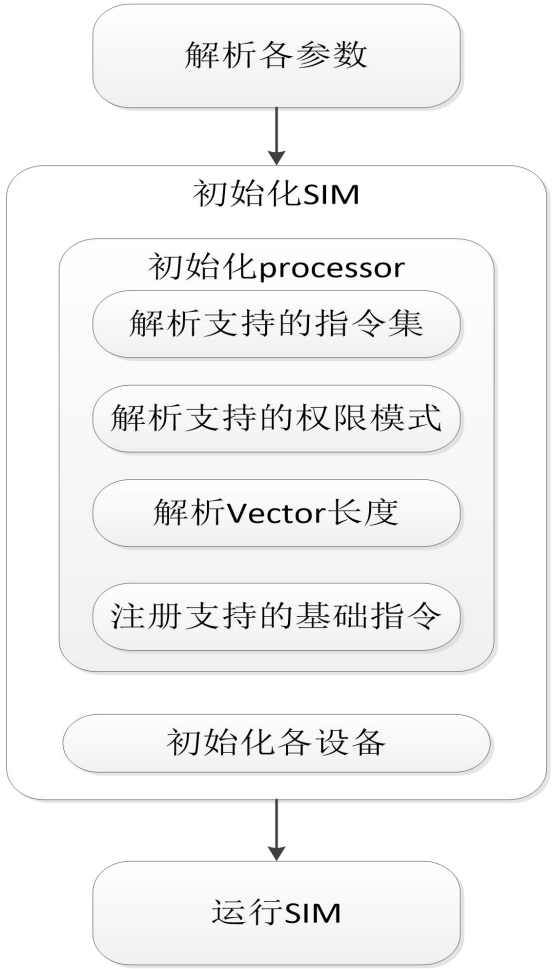
中国科学院软件研究所
PLCT实验室

李威威 liweiwei@iscas.ac.cn
王俊强 wangjunqiang@iscas.ac.cn
吴伟 wuwei2016@iscas.ac.cn

回顾

指令集模拟

- 内部指令集扩展模拟
- 外部指令集扩展模拟
  - Rocc扩展模拟



指令集和指令的关系

- 指令集包含指令

```
void processor_t::register_extension(extension_t* x)
{
 for (auto insn : x->get_instructions())
  register_insn(insn);
 build_opcode_map();

 if (disassembler)
  for (auto disasm_insn : x->get_disasms())
   disassembler->add_insn(disasm_insn);

 if (!custom_extensions.insert(std::make_pair(x->name(), x)).second) {
  fprintf(stderr, "extensions must have unique names (got two named
\"%s\"!)\n", x->name());
  abort();
 }

 x->set_processor(this);
}
```

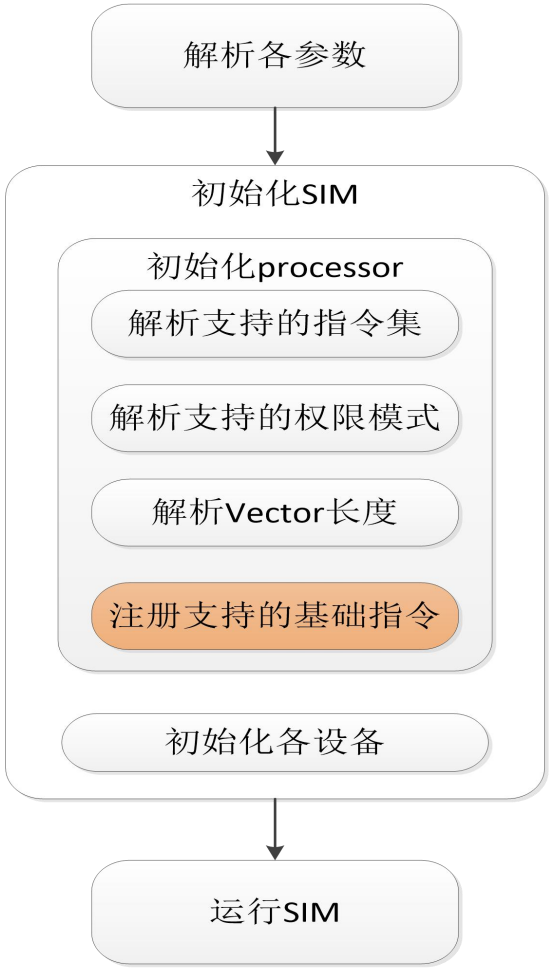
- 指令属于指令集

```
require_extension('C');
require(insn.rvc_rs2() != 0);
WRITE_RD(sext_xlen(RVC_RS1 + RVC_RS2));
```

```
#define require_extension(s) require(p->supports_extension(s))
```

```
bool supports_extension(unsigned char ext) {
 if (ext >= 'A' && ext <= 'Z')
  return ((state.misa >> (ext - 'A')) & 1);
 else
  return extension_table[ext];
}
```

基础指令注册




```
void processor_t::register_base_instructions()
{
 #define DECLARE_INSN(name, match, mask) \
   insn_bits_t name##_match = (match), name##_mask = (mask);
   unsigned name##_arch_en = (unsigned)-1;
 #define DECLARE_RV32_ONLY(name) {name##_arch_en = 32;}
 #define DECLARE_RV64_ONLY(name) {name##_arch_en = 64;}
 #include "encoding.h"
 #undef DECLARE_RV64_INSN
 #undef DECLARE_RV32_INSN
 #undef DECLARE_INSN

 #define DEFINE_INSN(name) \
   extern reg_t rv32_##name(processor_t*, insn_t, reg_t); \
   extern reg_t rv64_##name(processor_t*, insn_t, reg_t); \
   register_insn((insn_desc_t){ \
     name##_match, \
     name##_mask, \
     (name##_arch_en & 32) ? rv32_##name : nullptr, \
     (name##_arch_en & 64) ? rv64_##name : nullptr});
 #include "insn_list.h"
 #undef DEFINE_INSN
 register_insn({0, 0, &illegal_instruction, &illegal_instruction});
 build_opcode_map();
}
```

encoding.h

编码信息(riscv/encoding.h)

```
//CSR域信息
#define MSTATUS_UIE         0x00000001
#define MSTATUS_SIE         0x00000002
#define MSTATUS_HIE         0x00000004

//指令编码
#define MATCH_SLLI 0x1013
#define MASK_SLLI  0xfc00707f

//指令声明
DECLARE_INSN(slli, MATCH_SLLI, MASK_SLLI)
DECLARE_INSN(srli, MATCH_SRLI, MASK_SRLI)

//CSR地址
#define CSR_FFLAGS 0x1
#define CSR_FRM 0x2

//CSR声明
DECLARE_CSR(fflags, CSR_FFLAGS)
DECLARE_CSR(frm, CSR_FRM)
```

可通过riscv-opcode工具生成

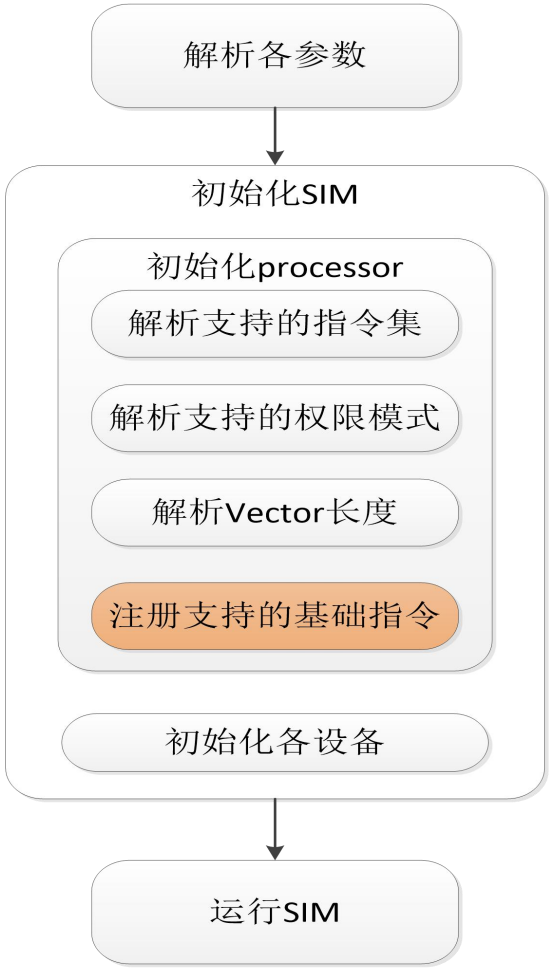
DEFINE_INSN生成

指令列表(riscv/riscv.mk.in)

```
riscv_insn_ext_i = \
        add \
        addi \
        addiw \
        addw \
        and \
        ...
riscv_insn_list = \
        $(riscv_insn_ext_a) \
        $(riscv_insn_ext_c) \
        $(riscv_insn_ext_i) \
        $(riscv_insn_ext_m) \
        $(riscv_insn_ext_f) \
        $(riscv_insn_ext_d) \
        $(riscv_insn_ext_zfh) \
        ...
```

```
insn_list.h: $(src_dir)/riscv/riscv.mk.in
        for insn in $(foreach
insn,$(riscv_insn_list),$(subst .,_,$(insn))) ; do \
                printf 'DEFINE_INSN(%s)\n'
"$${insn}" ; \
        done > $@.tmp
        mv $@.tmp $@
```

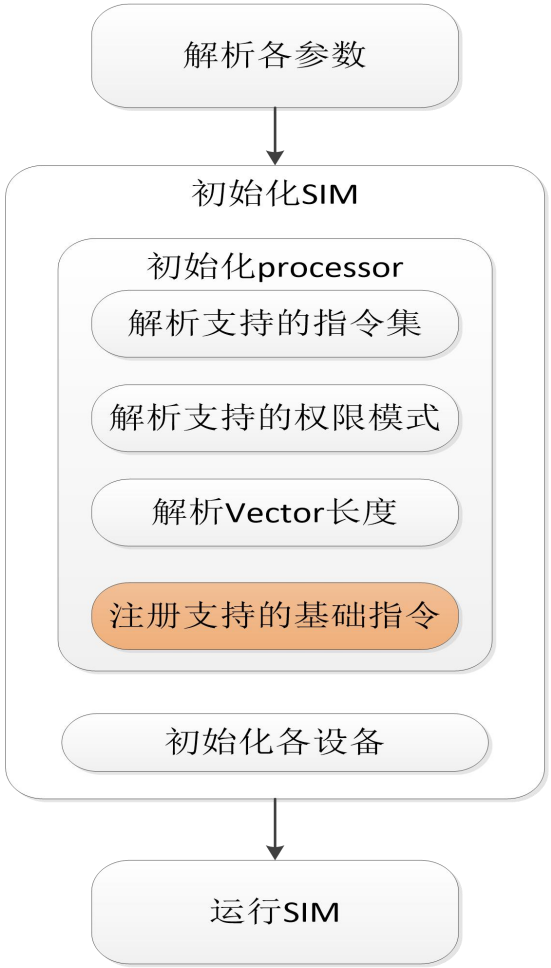
基础指令注册




```
void processor_t::register_base_instructions()
{
 #define DECLARE_INSN(name, match, mask) \
   insn_bits_t name##_match = (match), name##_mask = (mask);
   unsigned name##_arch_en = (unsigned)-1;
 #define DECLARE_RV32_ONLY(name) {name##_arch_en = 32;}
 #define DECLARE_RV64_ONLY(name) {name##_arch_en = 64;}
 #include "encoding.h"
 #undef DECLARE_RV64_INSN
 #undef DECLARE_RV32_INSN
 #undef DECLARE_INSN

 #define DEFINE_INSN(name) \
   extern reg_t rv32_##name(processor_t*, insn_t, reg_t); \
   extern reg_t rv64_##name(processor_t*, insn_t, reg_t); \
   register_insn((insn_desc_t){ \
     name##_match, \
     name##_mask, \
     (name##_arch_en & 32) ? rv32_##name : nullptr, \
     (name##_arch_en & 64) ? rv64_##name : nullptr});
 #include "insn_list.h"
 #undef DEFINE_INSN
 register_insn({0, 0, &illegal_instruction, &illegal_instruction});
 build_opcode_map();
}
```

基础指令注册




```
void processor_t::register_base_instructions()
{
 #define DECLARE_INSN(name, match, mask) \
   insn_bits_t name##_match = (match), name##_mask = (mask); \
   unsigned name##_arch_en = (unsigned)-1;
 #define DECLARE_RV32_ONLY(name) {name##_arch_en = 32;}
 #define DECLARE_RV64_ONLY(name) {name##_arch_en = 64;}
 #include "encoding.h"
 #undef DECLARE_RV64_INSN
 #undef DECLARE_RV32_INSN
 #undef DECLARE_INSN

 #define DEFINE_INSN(name) \
   extern reg_t rv32_##name(processor_t*, insn_t, reg_t); \
   extern reg_t rv64_##name(processor_t*, insn_t, reg_t); \
   register_insn((insn_desc_t){ \
     name##_match, \
     name##_mask, \
     (name##_arch_en & 32) ? rv32_##name : nullptr, \
     (name##_arch_en & 64) ? rv64_##name : nullptr});
 #include "insn_list.h"
 #undef DEFINE_INSN
 register_insn({0, 0, &illegal_instruction, &illegal_instruction});
 build_opcode_map();
}
```

```
void processor_t::register_insn(insn_desc_t desc)
{
  instructions.push_back(desc);
}
```

```
struct insn_desc_t
{
  insn_bits_t match;
  insn_bits_t mask;
  insn_func_t rv32;
  insn_func_t rv64;
};
```

基础指令功能模板

riscv/insn_template.c

```
reg_t rv32_NAME(processor_t* p, insn_t insn, reg_t pc)
{
  int xlen = 32;
  reg_t npc = sext_xlen(pc + insn_length(OPCODE));
  #include "insns/NAME.h"
  trace_opcode(p, OPCODE, insn);
  return npc;
}

reg_t rv64_NAME(processor_t* p, insn_t insn, reg_t pc)
{
  int xlen = 64;
  reg_t npc = sext_xlen(pc + insn_length(OPCODE));
  #include "insns/NAME.h"
  trace_opcode(p, OPCODE, insn);
  return npc;
}
```

基础指令功能函数生成

riscv/insn_template.c

```
reg_t rv32_NAME(processor_t* p, insn_t insn, reg_t pc)
{
  int xlen = 32;
  reg_t npc = sext_xlen(pc + insn_length(OPCODE));
  #include "insns/NAME.h"
  trace_opcode(p, OPCODE, insn);
  return npc;
}

reg_t rv64_NAME(processor_t* p, insn_t insn, reg_t pc)
{
  int xlen = 64;
  reg_t npc = sext_xlen(pc + insn_length(OPCODE));
  #include "insns/NAME.h"
  trace_opcode(p, OPCODE, insn);
  return npc;
}
```

生成 <name>.cc

```
riscv_gen_srcs = \
        $(addsuffix .cc,$(riscv_insn_list))

$(riscv_gen_srcs): %.cc: insns/%.h insn_template.cc
        sed 's/NAME/$(subst .cc,,$@)/'
$(src_dir)/riscv/insn_template.cc | sed
's/OPCODE/$(call
get_opcode,$(src_dir)/riscv/encoding.h,$(subst .cc,,$@))
/' > $@
```

基础指令功能代码

riscv/insn_template.c

```
reg_t rv32_NAME(processor_t* p, insn_t insn, reg_t pc)
{
  int xlen = 32;
  reg_t npc = sext_xlen(pc + insn_length(OPCODE));
  #include "insns/NAME.h"
  trace_opcode(p, OPCODE, insn);
  return npc;
}

reg_t rv64_NAME(processor_t* p, insn_t insn, reg_t pc)
{
  int xlen = 64;
  reg_t npc = sext_xlen(pc + insn_length(OPCODE));
  #include "insns/NAME.h"
  trace_opcode(p, OPCODE, insn);
  return npc;
}
```

- 指令功能代码：**riscv/insns/<name>.h**
- add.h

```
WRITE_RD(sext_xlen(RS1 + RS2));
```

- riscv/decode.h

```
#define WRITE_RD(value) WRITE_REG(insn.rd(), value)

#define WRITE_REG(reg, value) STATE.XPR.write(reg, value)

#define RS1 READ_REG(insn.rs1())
#define RS2 READ_REG(insn.rs2())
#define RS3 READ_REG(insn.rs3())

#define READ_REG(reg) STATE.XPR[reg]
```

指令所能访问的资源

riscv/insn_template.c

```
reg_t rv32_NAME(processor_t* p, insn_t insn, reg_t pc)
reg_t rv64_NAME(processor_t* p, insn_t insn, reg_t pc)
```

```
struct state_t
{
  ...
  reg_t pc;
  regfile_t<reg_t, NXPR, true> XPR;
  regfile_t<freg_t, NFPR, false> FPR;
  ...
  reg_t misa;
  reg_t mstatus;
  reg_t mepc;
  reg_t mtval;
  reg_t mscratch;
  reg_t mtvec;
  reg_t mcause;
  reg_t minstret;
  reg_t mie;
  reg_t mip;
  ...
}
```

```
class vectorUnit_t {
   public:
     processor_t* p;
     void *reg_file;
     char reg_referenced[NVPR];
     int setvl_count;
     reg_t vlmax;
     reg_t vstart, vxrm, vxsat, vl, vtype, vlenb;
     reg_t vma, vta;
     reg_t vsew;
     float vflmul;
     reg_t ELEN, VLEN;
     bool vill;
     bool vstart_alu;
    ...
}
```

CSR访问

```
void processor_t::set_csr(int which, reg_t val)
{
...
  switch (which)
  {
    case CSR_MENTROPY:
      es.set_mentropy(val);
      break;
    case CSR_MNOISE:
      es.set_mnoise(val);
      break;
    case CSR_FFLAGS:
      dirty_fp_state;
      state.fflags = val & (FSR_AEXC >>
FSR_AEXC_SHIFT);
      break;
      ...
  }
 }
```

```
reg_t processor_t::get_csr(int which, insn_t insn, bool write, bool peek)
{
  ...
  switch (which)
  {
    case CSR_MENTROPY:
      if(!supports_extension('K'))
          break;
      return es.get_mentropy();
    case CSR_MNOISE:
      if(!supports_extension('K'))
          break;
      return es.get_mnoise();
    case CSR_FFLAGS:
      require_fp;
      if (!supports_extension('F'))
        break;
      ret(state.fflags);
      ...
    }
}
```

指令执行流程

- 指令执行流程： riscv/execute.cc：step()

```
auto ic_entry = _mmu->access_icache(pc);

#define ICACHE_ACCESS(i) { \
insn_fetch_t fetch = ic_entry->data; \
pc = execute_insn(this, pc, fetch); \
ic_entry = ic_entry->next; \
if (i == mmu_t::ICACHE_ENTRIES-1) break; \
if (unlikely(ic_entry->tag != pc)) break; \
if (unlikely(instret+1 == n)) break; \
  instret++; \
  state.pc = pc; \
}
```

```
static reg_t execute_insn(processor_t* p, reg_t pc,
insn_fetch_t fetch)
{
 ...
  npc = fetch.func(p, fetch.insn, pc);
 …
}
```

```
struct insn_fetch_t
{
 insn_func_t func;
 insn_t insn;
};
```

- riscv/mmu.h

```
inline icache_entry_t* access_icache(reg_t addr)
{
 icache_entry_t* entry = &icache[icache_index(addr)];
 if (likely(entry->tag == addr))
   return entry;
 return refill_icache(addr, entry);
}
```

```
inline icache_entry_t* refill_icache(reg_t addr, icache_entry_t* entry)
{
 auto tlb_entry = translate_insn_addr(addr);
 insn_bits_t insn = from_le(*(uint16_t*)(tlb_entry.host_offset + addr));
 int length = insn_length(insn);
  ...
 insn_fetch_t fetch = {proc->decode_insn(insn), insn};
 entry->tag = addr;
 entry->next = &icache[icache_index(addr + length)];
 entry->data = fetch;
 ...
 return entry;
}
```

指令解码

riscv/processor.cc

```
insn_func_t processor_t::decode_insn(insn_t insn)
{
  // look up opcode in hash table
  size_t idx = insn.bits() % OPCODE_CACHE_SIZE;
  insn_desc_t desc = opcode_cache[idx];

  if (unlikely(insn.bits() != desc.match || !(xlen == 64 ?
desc.rv64 : desc.rv32))) {
    // fall back to linear search
    int cnt = 0;
    insn_desc_t* p = &instructions[0];
    while ((insn.bits() & p->mask) != p->match || !(xlen ==
64 ? desc.rv64 : desc.rv32))
      p++, cnt++;
    desc = *p;
```

```
  if (p->mask != 0 && p > &instructions[0]) {
    if (p->match != (p-1)->match && p->match != (p+1)->match) {
      // move to front of opcode list to reduce miss penalty
      while (--p >= &instructions[0])
        *(p+1) = *p;
      instructions[0] = desc;
    }
  }

  opcode_cache[idx] = desc;
  opcode_cache[idx].match = insn.bits();
 }

 return xlen == 64 ? desc.rv64 : desc.rv32;
}
```

基础指令添加过程

第一步：
指令功能

第二步：
指令编码

第三步：
注册指令

第四步：
重编译

第五步：
指令测试

基础指令添加举例—添加指令功能

以添加NICE demo指令为例

第二步：
指令编码

第四步：
重编译

第一步：
指令功能

第三步：
注册指令

第五步：
指令测试

添加custom_rowsum/wsetup/sbuf指令功能代码，
创建riscv/insns/custom_rowsum/wsetup/sbuf.h：

```
require_extension(EXT_ZNICE);
reg_t xs1 = RS1;
reg_t rowsum = 0;
reg_t data;
for (reg_t i = 0; i <= p->rocc.matrix_config; i++) {
  data = MMU.load_uint32(xs1 + 4 * i);
  rowsum += data;
  p->rocc.row_buffer[i] += data;
}
WRITE_RD(rowsum);
```

基础指令添加举例—扩展processor状态

以添加NICE demo指令为例

第二步：
指令编码

第四步：
重编译

第一步：
指令功能

第三步：
注册指令

第五步：
指令测试

```
class processor_t : public
abstract_device_t
{
  …
  vectorUnit_t VU;
  rocc_state_t rocc;
};
```

```
struct rocc_state_t
{
reg_t matrix_config;
reg_t row_buffer[MAX_COL_NUM];
};
```

基础指令添加举例—添加指令编码

以添加NICE demo指令为例

第二步：指令编码

第一步：指令功能

第三步：注册指令

第四步：重编译

第五步：指令测试

添加NICE demo编码及声明到riscv/encoding.h

```
#define MATCH_CUSTOM_SBUF 0x400207b
#define MASK_CUSTOM_SBUF  0xfe00707f
#define MATCH_CUSTOM_WSETUP 0x800207b
#define MASK_CUSTOM_WSETUP  0xfe00707f
#define MATCH_CUSTOM_ROWSUM 0xc00607b
#define MASK_CUSTOM_ROWSUM  0xfe00707f
```

```
DECLARE_RV32_ONLY
DECLARE_RV64_ONLY
```

```
DECLARE_INSN(custom_sbuf, MATCH_CUSTOM_SBUF, MASK_CUSTOM_SBUF)
DECLARE_INSN(custom_wsetup, MATCH_CUSTOM_WSETUP, MASK_CUSTOM_WSETUP)
DECLARE_INSN(custom_rowsum, MATCH_CUSTOM_ROWSUM, MASK_CUSTOM_ROWSUM)
```

基础指令添加举例—添加指令编码

以添加NICE demo指令为例

第二步：指令编码

第一步：指令功能

第三步：注册指令

第四步：重编译

第五步：指令测试

通过riscv-opcodes生成riscv/encoding.h
opcode-custom中添加：

```
custom_sbuf     rd rs1 rs2 31..25=2 14..12=2 6..2=0x1E 1..0=3
custom_wsetup   rd rs1 rs2 31..25=4 14..12=2 6..2=0x1E 1..0=3
custom_rowsum   rd rs1 rs2 31..25=6 14..12=6 6..2=0x1E 1..0=3
```

基础指令添加举例—注册指令

以添加NICE demo指令为例

第一步：
指令功能

第二步：
指令编码

第三步：
注册指令

第四步：
重编译

第五步：
指令测试

将指令添加到riscv_insn_list中(riscv/riscv.mk.in)：

```
riscv_insn_nice = \
custom_sbuf \
custom_wsetup \
custom_rowsum \

riscv_insn_list = \
...
$(riscv_insn_ext_h) \
$(riscv_insn_priv) \
$(riscv_insn_nice) \
```

指令功能代码

custom_rowsum完整指令功能代码

```
#include "insn_template.h"

reg_t rv32_custom_rowsum(processor_t* p, insn_t insn, reg_t
pc)
{
  int xlen = 32;
  reg_t npc = sext_xlen(pc +
insn_length( MATCH_CUSTOM_ROWSUM));
  #include "insns/custom_rowsum.h"
  trace_opcode(p,  MATCH_CUSTOM_ROWSUM, insn);
  return npc;
}
```

```
reg_t rv64_custom_rowsum(processor_t* p, insn_t insn,
reg_t pc)
{
  int xlen = 64;
  reg_t npc = sext_xlen(pc +
insn_length( MATCH_CUSTOM_ROWSUM));
  #include "insns/custom_rowsum.h"
  trace_opcode(p,  MATCH_CUSTOM_ROWSUM, insn);
  return npc;
}
```

基础指令添加举例—重编译

以添加NICE demo指令为例

第一步：
指令功能

第二步：
指令编码

第三步：
注册指令

第四步：
重编译

第五步：
指令测试

重新编译安装spike：

```
cd "${RISCV}/riscv-isa-sim/build"
make
sudo make install
```

基础指令添加举例—测试

以添加NICE demo指令为例

第一步：
指令功能

第二步：
指令编码

第三步：
注册指令

第四步：
重编译

第五步：
指令测试

添加新指令测试程序nicext_test.c：

```
int main(void)
{
 ...
  printf("2. Do reference matrix column sum and row sum\r\n");
  normal_case(array, col_sum_ref, row_sum_ref);
  printf("2. Do nice matrix column sum and row sum\r\n");
  nice_case(array, col_sum_nice, row_sum_nice);
  ...
  if (compare_result(col_sum_ref, row_sum_ref, col_sum_nice, row_sum_nice) == 0) {
    printf("PASS\r\n");
  } else {
    printf("FAIL\r\n");
  }
  return 0;
}
```

基础指令添加举例—测试

以添加NICE demo指令为例

第一步：
指令功能

第二步：
指令编码

第三步：
注册指令

第四步：
重编译

第五步：
指令测试

添加新指令测试程序nicext_test.c：

```
void custom_sbuf(unsigned long * addr)
{
  int zero = 0;
  asm volatile(".insn r 0x7b, 2, 2, x0, %1, x0" : "=r"(zero) : "r"(addr));
}
```

```
int main(void)
{
 ...
  printf("2. Do reference matrix column sum and row sum\r\n");
  normal_case(array, col_sum_ref, row_sum_ref);
  printf("2. Do nice matrix column sum and row sum\r\n");
  nice_case(array, col_sum_nice, row_sum_nice);
  ...
  if (compare_result(col_sum_ref, row_sum_ref, col_sum_nice, row_sum_nice) == 0) {
    printf("PASS\r\n");
  } else {
    printf("FAIL\r\n");
  }
  return 0;
}
```

基础指令添加举例—测试

第一步：
指令功能

第二步：
指令编码

第三步：
注册指令

第四步：
重编译

第五步：
指令测试

测试程序编译运行：

- 编译

  riscv64-unknown-elf-gcc nicext_test.c -o test

- 运行

  spike --isa=rv64gc_znice pk test

基础指令添加举例—测试结果

第一步：
指令功能

第二步：
指令编码

第三步：
注册指令

第四步：
重编译

第五步：
指令测试

```
liww@liww-tm:build$ ./spike --isa=rv64gc_znice /opt/riscv/bin/pk ../nicext/test
bbl loader

Nuclei Nice Acceleration Demonstration
1. Print input matrix array
the element of array is :
        10      30      90
        20      40      80
        30      90      120

2. Do reference matrix column sum and row sum
2. Do nice matrix column sum and row sum
3. Compare reference and nice result
  1) Reference result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
  2) Nice result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
  3) Compare reference vs nice: PASS
liww@liww-tm:build$
```

外部扩展指令添加

```
class nice_demo_t : public extension_t
{
 public:
  const char* name() { return "nice_demo"; }

  nice_demo_t() {}

  std::vector<insn_desc_t> get_instructions() {
    std::vector<insn_desc_t> insns;
    insns.push_back((insn_desc_t){0x0400207b, 0xFE00707F, custom_sbuf, custom_sbuf});
    insns.push_back((insn_desc_t){0x0800207b, 0xFE00707F, custom_wsetup, custom_wsetup});
    insns.push_back((insn_desc_t){0x0C00607b, 0xFE00707F, custom_rowsum, custom_rowsum});
    return insns;
  }

  std::vector<disasm_insn_t*> get_disasms() {
    std::vector<disasm_insn_t*> insns;
    insns.push_back(new disasm_insn_t("custom_sbuf", 0x0400207b, 0xFE00707F, {&xrs1}));
    insns.push_back(new disasm_insn_t("custom_wsetup", 0x0800207b, 0xFE00707F, {&xrs1}));
    insns.push_back(new disasm_insn_t("custom_rowsum", 0x0C00607b, 0xFE00707F, {&xrd,
&xrs1}));
    return insns;
  }
};
```

```
static reg_t matrix_config;
static reg_t row_buffer[MAX_COL_NUM];
```

```
static reg_t custom_sbuf(processor_t* p,
insn_t insn, reg_t pc)
{
  reg_t xs1 = RS1;
  for (reg_t i = 0; i <= matrix_config; i++) {
    p->get_mmu()->store_uint32(xs1 + 4 * i,
row_buffer[i]);
  }
  return pc + 4;
}
```

外部扩展指令添加—测试结果

第一步：
指令功能

第二步：
指令编码

第三步：
注册指令

第四步：
重编译

第五步：
指令测试

```
liww@liww-tm:build$ ./spike --isa=rv64gc_xnice_demo --extlib=./libnicext.so /opt/riscv/bin/pk
 ../nicext/test
bbl loader

Nuclei Nice Acceleration Demonstration
1. Print input matrix array
the element of array is :
        10      30      90
        20      40      80
        30      90      120

2. Do reference matrix column sum and row sum
2. Do nice matrix column sum and row sum
3. Compare reference and nice result
  1) Reference result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
  2) Nice result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
  3) Compare reference vs nice: PASS
```

外部扩展指令添加——Rocc

```
class rocc_t : public extension_t
{
 public:
  virtual reg_t custom0(rocc_insn_t insn, reg_t xs1, reg_t xs2);
  virtual reg_t custom1(rocc_insn_t insn, reg_t xs1, reg_t xs2);
  virtual reg_t custom2(rocc_insn_t insn, reg_t xs1, reg_t xs2);
  virtual reg_t custom3(rocc_insn_t insn, reg_t xs1, reg_t xs2);
  std::vector<insn_desc_t> get_instructions();
  std::vector<disasm_insn_t*> get_disasms();
};
```

```
class nice_rocc_demo_t : public rocc_t
{
  ...
 private:
  reg_t matrix_config;
  reg_t row_buffer[MAX_COL_NUM];
};
```

```
reg_t custom3(rocc_insn_t insn, reg_t xs1, reg_t xs2)
 {
  reg_t rowsum = 0, data;

  switch (insn.funct)
  {
   case 2: // custom_sbuf
    for (reg_t i = 0; i <= matrix_config; i++) {
     p->get_mmu()->store_uint32(xs1 + 4 * i, row_buffer[i]);
    }
    break;
   case 4: // custom_wsetup
    ...
    break;
   case 6: // custom_rowsum
    ...
    break;
   default:
    illegal_instruction();
    break;
  }

  return rowsum;
 }
```

外部扩展指令添加—Rocc测试结果

第一步：
指令功能

第二步：
指令编码

第三步：
注册指令

第四步：
重编译

第五步：
指令测试

```
liww@liww-tm:build$ ./spike --isa=rv64gc_xnice_rocc_demo --extlib=./libnicext.so /opt/riscv/b
in/pk ../nicext/test
bbl loader

Nuclei Nice Acceleration Demonstration
1. Print input matrix array
the element of array is :
        10      30      90
        20      40      80
        30      90      120

2. Do reference matrix column sum and row sum
2. Do nice matrix column sum and row sum
3. Compare reference and nice result
  1) Reference result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
  2) Nice result:
the sum of each row is :
                130     140     240
the sum of each col is :
                60      160     290
  3) Compare reference vs nice: PASS
```

CSR添加过程

第一步：
添加CSR编码　　riscv/encoding.h　CSR_XXX

第二步：
添加CSR状态　　riscv/processor.h:　state_t/processor_t

第三步：
添加csr读写操作　　riscv/processor.cc:　set/get_csr() -> switch()

谢谢各位

欢迎提问、讨论、交流合作