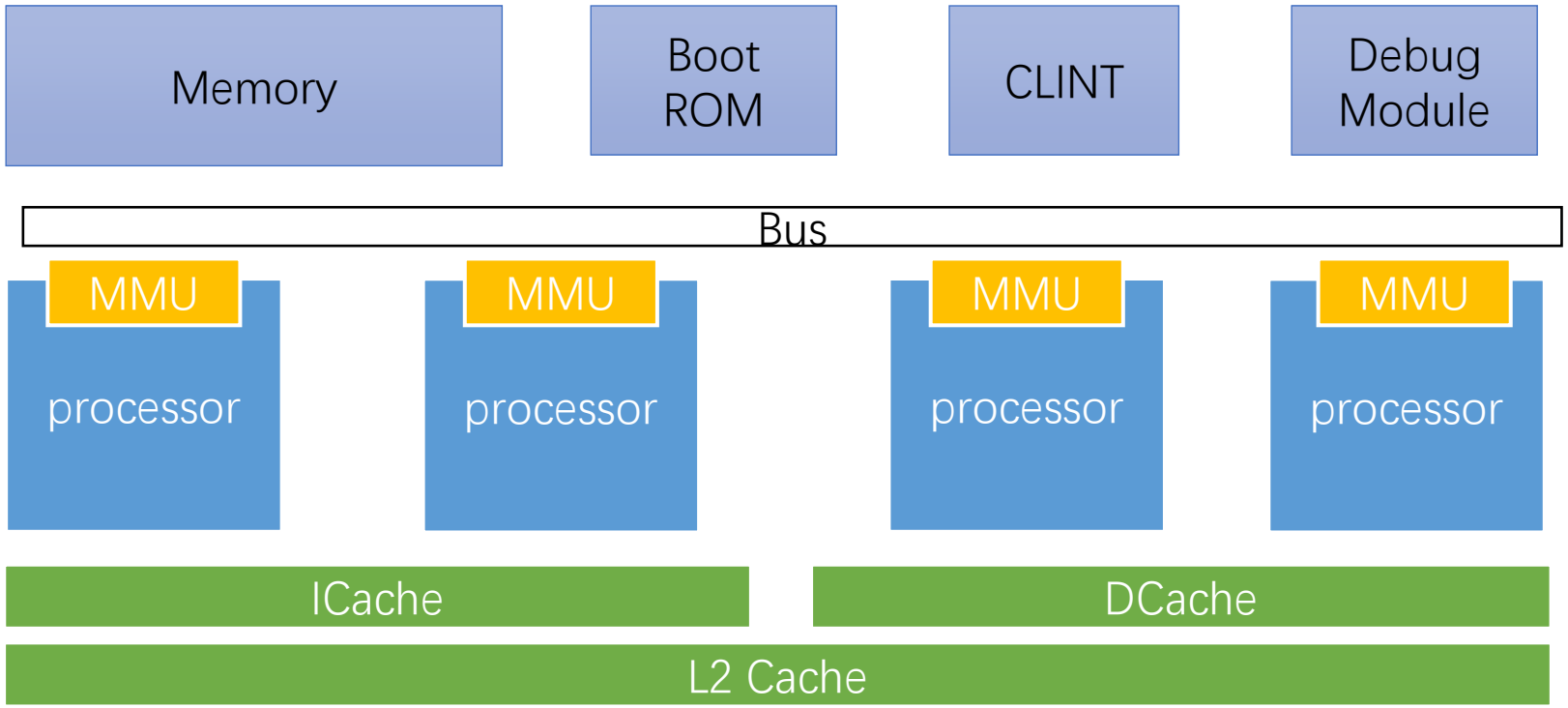
Spike扩展方式简介

李威威
PLCT实验室

目 录

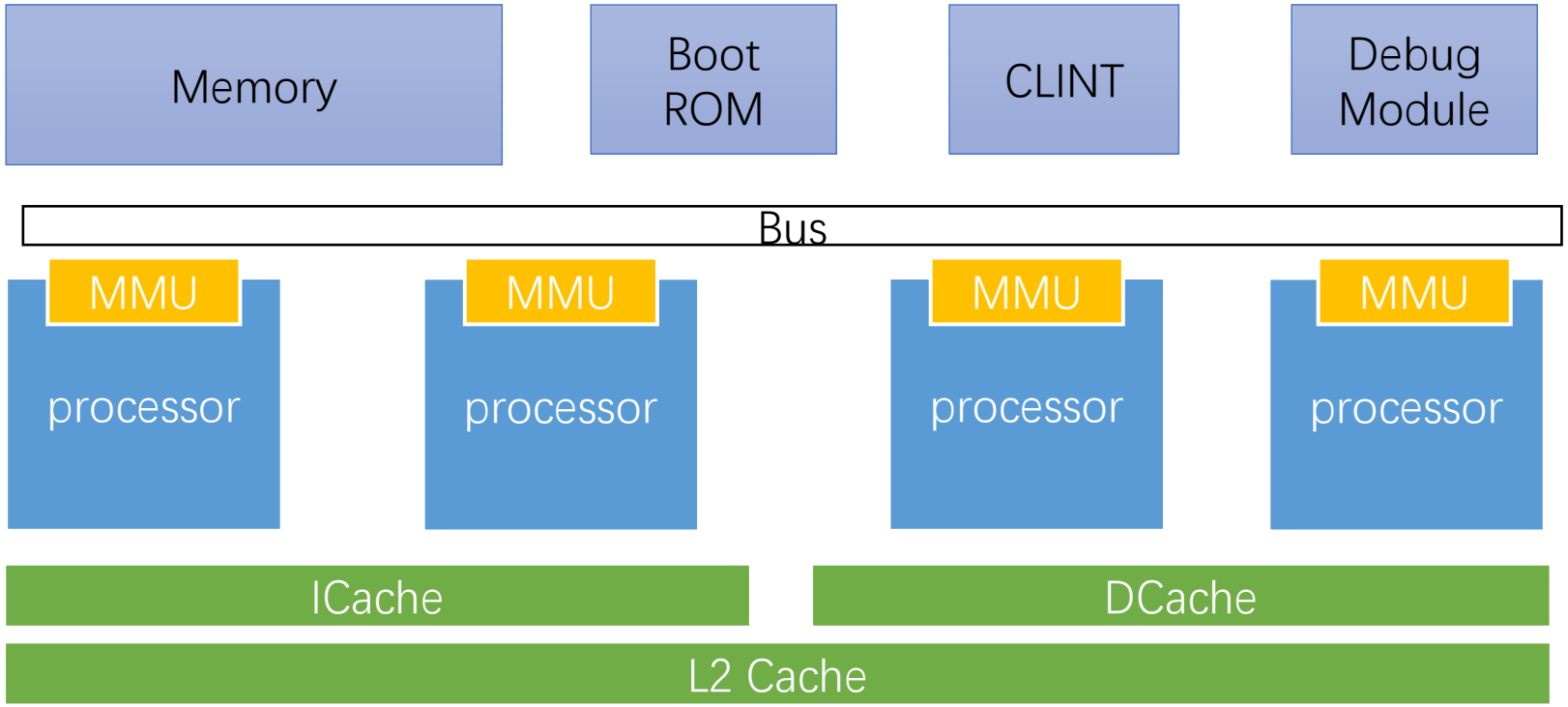
Spike简介

Spike是针对RISCV的轻量级指令集模拟器



Spike简介

Spike是针对RISCV的轻量级指令集模拟器



PLCT主要工作：

1.新特性支持：snapshot

2.新设备支持：块设备

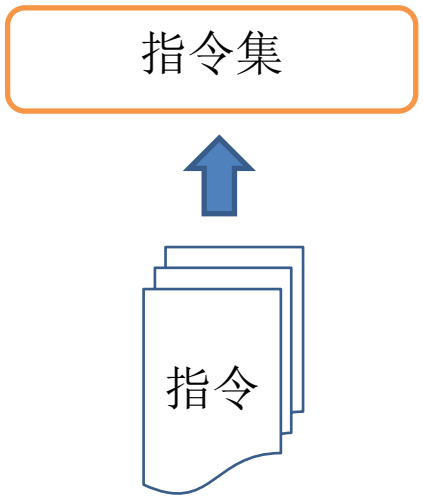
3.新指令集扩展支持：Zfinx支持

Github 仓库：https://github.com/isrc-cas/plct-spike

目录

Spike指令（集）扩展

Spike指令（集）扩展




```
std::vector<bool> extension_table;
```

```
std::vector<insn_desc_t> instructions;
```

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

fadd_s:

```
require_extension('F');
require_fp;
softfloat_roundingMode = RM;
WRITE_FRD(f32_add(f32(FRS1), f32(FRS2)));
set_fp_exceptions;
```

Spike指令（集）扩展

指令集支持

- 指令集选项：--isa

默认支持：RV64IMAFDC

riscv/processor.cc：**void processor_t::parse_isa_string(const char*** str)

  - 普通指令集
  - 固有扩展指令集
  - 外部扩展指令集

Spike指令（集）扩展

指令集支持

- 指令集选项：--isa

  riscv/processor.cc：**void processor_t::parse_isa_string(const char*** str)

  - 普通指令集：单字符
  - 固有扩展指令集
  - 外部扩展指令集

```
 const char* all_subsets = "imafdqcbh"
#ifdef __SIZEOF_INT128__
   "v"
#endif
   "";
```

```
if (islower(*p)) {
     max_isa |= 1L << (*p - 'a');
     extension_table[toupper(*p)] = true;

     if (strchr(all_subsets, *p)) {
       p++;
     }
   ...
}
```

Spike指令（集）扩展

指令集支持

- 指令集选项： --isa

  riscv/processor.cc： **void processor_t::parse_isa_string(const char*** str)

  - 普通指令集
  - 固有扩展指令集：_<name>
  - 外部扩展指令集

```
if (*p == '_') {
    const char* ext = p + 1, *end = ext;
    if (*ext == 'x') {
      p++;
      continue;
    }
    while (islower(*end))
      end++;
    auto ext_str = std::string(ext, end - ext);
    printf("-ext:%s\n", ext_str.c_str());
    if (ext_str == "zfh") {
      extension_table[EXT_ZFH] = true;
    }
```

Spike指令（集）扩展

指令集支持

- 指令集选项： --isa

  riscv/processor.cc： **void processor_t::parse_isa_string(const char* str)**

  - 普通指令集
  - 固有扩展指令集
  - 外部扩展指令集:x<name>

```
if (*p == 'x') {
  const char* ext = p + 1, *end = ext;
  while (islower(*end) || *end == '_')
    end++;

  auto ext_str = std::string(ext, end - ext);
  if (ext_str != "dummy")
    register_extension(find_extension(ext_str.c_str())());
  p = end;
}
```

Spike指令（集）扩展

```
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 (ext != NULL)
    throw std::logic_error("only one extension may be registered");
  ext = x;
  x->set_processor(this);
}
```

```
std::vector<insn_desc_t> instructions;
```

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

Spike指令（集）扩展

customext/cflush.c

```
class cflush_t : public extension_t
{
...
std::vector<insn_desc_t> get_instructions() {
    std::vector<insn_desc_t> insns;
    insns.push_back((insn_desc_t){0xFC000073, 0xFFF07FFF, custom_cflush, custom_cflush});
  ...  return insns;
  }

  std::vector<disasm_insn_t*> get_disasms() {
    std::vector<disasm_insn_t*> insns;
    insns.push_back(new disasm_insn_t("cflush.d.l1", 0xFC000073, 0xFFF07FFF, {&xrs1}));
...  return insns;
  }
}
REGISTER_EXTENSION(cflush, []() { return new cflush_t; })
```

```
#define REGISTER_EXTENSION(name, constructor) \
  class register_##name { \
    public: register_##name() { register_extension(#name, constructor); } \
  }; static register_##name dummy_##name;
```

Spike指令（集）扩展

指令支持

- 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);
  #include "encoding.h"
  #undef DECLARE_INSN

  #define DEFINE_INSN(name) \
    REGISTER_INSN(this, name, name##_match, name##_mask)
  #include "insn_list.h"
  #undef DEFINE_INSN

  register_insn({0, 0, &illegal_instruction, &illegal_instruction});
  build_opcode_map();
}
```

Spike指令（集）扩展

指令支持

- 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);
 #include "encoding.h"
 #undef DECLARE_INSN

 #define DEFINE_INSN(name) \
   REGISTER_INSN(this, name, name##_match, name##_mask)
 #include "insn_list.h"
 #undef DEFINE_INSN

 register_insn({0, 0, &illegal_instruction, &illegal_instruction});
 build_opcode_map();
}
```

```
#define MATCH_BEQ 0x63
#define MASK_BEQ  0x707f
#define MATCH_BNE 0x1063
#define MASK_BNE  0x707f
...
DECLARE_INSN(beq, MATCH_BEQ, MASK_BEQ)
DECLARE_INSN(bne, MATCH_BNE, MASK_BNE)
```

Spike指令（集）扩展

指令支持

- 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);
 #include "encoding.h"
 #undef DECLARE_INSN

 #define DEFINE_INSN(name) \
   REGISTER_INSN(this, name, name##_match, name##_mask)
 #include "insn_list.h"
 #undef DEFINE_INSN

 register_insn({0, 0, &illegal_instruction, &illegal_instruction});
 build_opcode_map();
}
```

```
#define REGISTER_INSN(proc, name, match, mask) \
  extern reg_t rv32_##name(processor_t*, insn_t, reg_t); \
  extern reg_t rv64_##name(processor_t*, insn_t, reg_t); \
  proc->register_insn((insn_desc_t){match, mask,
rv32_##name, rv64_##name});
```

```
DEFINE_INSN(amoadd_d)
DEFINE_INSN(amoadd_w)
DEFINE_INSN(amoand_d)
DEFINE_INSN(amoand_w)
```

Spike指令（集）扩展

指令支持

- 指令功能模板：riscv/insn_template.c

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

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

  - riscv/decode.h

```
#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]
```

```
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;
}
```

Spike指令（集）扩展

指令支持

- 指令执行流程： riscv/execute.cc
- step()

```
insn_fetch_t fetch = mmu->load_insn(pc);
if (debug && !state.serialized)
  disasm(fetch.insn);
pc = execute_insn(this, pc, fetch);
advance_pc();
```

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

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

Spike指令（集）扩展

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)) {
    // fall back to linear search
    insn_desc_t* p = &instructions[0];
    while ((insn.bits() & p->mask) != p->match)
      p++;
    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();
  }
```

Spike指令（集）扩展

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)) {
    // fall back to linear search
    insn_desc_t* p = &instructions[0];
    while ((insn.bits() & p->mask) != p->match)
      p++;
    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();
  }
```

Spike指令（集）扩展

以zfinx扩展为例

- 添加新的指令集标志，类似于zfh： **parse_isa_string**

```
if (ext_str == "zfinx") {
    extension_table[EXT_ZFINX] = true;
    max_isa |= 1L << ('f' - 'a');
 }
```

```
regfile_t<reg_t, NXPR, true> XPR;
regfile_t<freg_t, NFPR, false> FPR;
```

- 修改浮点指令功能实现，以fadd_s为例

```
require_extension('F');
require_fp;
softfloat_roundingMode = RM;
WRITE_FRD(f32_add(f32(FRS1), f32(FRS2)));
set_fp_exceptions;
```

```
#define FRS1 READ_FREG(insn.rs1())
#define FRS2 READ_FREG(insn.rs2())
#define READ_FREG(reg) STATE.FPR[reg]
```

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

目录

Spike设备扩展

已支持的设备类型

riscv/devices.h

```
class bus_t : public abstract_device_t
class rom_device_t : public abstract_device_t
class mem_t : public abstract_device_t
class clint_t : public abstract_device_t
class debug_module_t : public abstract_device_t
class mmio_plugin_device_t : public abstract_device_t
```

Spike设备扩展

已支持的设备类型

riscv/devices.h

```
class bus_t : public abstract_device_t
 class rom_device_t : public abstract_device_t
 class mem_t : public abstract_device_t
 class clint_t : public abstract_device_t
 class debug_module_t : public abstract_device_t
class mmio_plugin_device_t : public abstract_device_t
```

```
class abstract_device_t {
 public:
  virtual bool load(reg_t addr, size_t len, uint8_t* bytes) = 0;
  virtual bool store(reg_t addr, size_t len, const uint8_t* bytes) =
0;
  virtual ~abstract_device_t() {}
};
```

Spike设备扩展

已支持的设备类型

Mmio plugin设备

riscv/devices.h

```
class mmio_plugin_device_t : public abstract_device_t
{
 public:
  mmio_plugin_device_t(const std::string& name,
const std::string& args);
  virtual ~mmio_plugin_device_t() override;

  virtual bool load(reg_t addr, size_t len, uint8_t* bytes)
override;
  virtual bool store(reg_t addr, size_t len, const uint8_t*
bytes) override;

 private:
  mmio_plugin_t plugin;
  void* user_data;
};
```

Spike设备扩展

plugin设备接口

riscv/mmio_plugin.h

```
typedef struct {
  // Allocate user data for an instance of the plugin.
  void* (*alloc)(const char*);

  // Load a memory address of the MMIO plugin. The parameters are the
user_data
  bool (*load)(void*, reg_t, size_t, uint8_t*);

  // Store some bytes to a memory address of the MMIO plugin. The parameters
are
  bool (*store)(void*, reg_t, size_t, const uint8_t*);

  // Deallocate the data allocated during the call to alloc. The parameter is a
  void (*dealloc)(void*);
} mmio_plugin_t;
```

Spike设备扩展

已支持的设备类型

Mmio plugin设备

riscv/devices.cc

```
mmio_plugin_device_t::mmio_plugin_device_t(const std::string& name,
                                           const std::string& args)
  : plugin(mmio_plugin_map().at(name)), user_data((*plugin.alloc)(args.c_str()))
{
}
mmio_plugin_device_t::~mmio_plugin_device_t()
{
  (*plugin.dealloc)(user_data);
}
bool mmio_plugin_device_t::load(reg_t addr, size_t len, uint8_t* bytes)
{
  return (*plugin.load)(user_data, addr, len, bytes);
}
bool mmio_plugin_device_t::store(reg_t addr, size_t len, const uint8_t* bytes)
{
  return (*plugin.store)(user_data, addr, len, bytes);
}
```

Spike设备扩展

已支持的设备类型

Mmio plugin设备注册

```
using mmio_plugin_map_t = std::map<std::string, mmio_plugin_t>;
```

riscv/devices.cc

Spike设备扩展

已支持的设备类型

Mmio plugin设备注册

```
using mmio_plugin_map_t = std::map<std::string, mmio_plugin_t>;
```

riscv/devices.cc

```
void register_mmio_plugin(const char* name_cstr,
                          const mmio_plugin_t* mmio_plugin)
{
  std::string name(name_cstr);
  if (!mmio_plugin_map().emplace(name, *mmio_plugin).second) {
    throw std::runtime_error("Plugin \"" + name + "\" already registered!");
  }
}
```

Spike设备扩展

以块设备功能实现

```
void* test_mmio_plugin_alloc(const char* args)
{
    printf("ALLOC -- ARGS=%s\n", args);
    int fd = open(args, O_RDWR, 0);
    ....
    struct stat st; //定义文件信息结构体
    int r=fstat(fd,&st);
    ...
    void *
p=mmap(NULL,len,PROT_READ|PROT_WRITE,MAP_S
HARED,fd,0);
    return p;
}
```

```
bool test_mmio_plugin_load(void* self, reg_t addr, size_t len,
uint8_t* bytes)
{
    memcpy(bytes, (char *)self + addr, len);
    return true;
}
```

```
bool test_mmio_plugin_store(void* self, reg_t addr, size_t
len, const uint8_t* bytes)
{
    memcpy((char *)self + addr, bytes, len);
    msync((char *)self + addr,len,0);
    return true;
}
```

```
void test_mmio_plugin_dealloc(void* self)
{
    munmap(self, len);
    printf("DEALLOC -- SELF=%p\n", self);
}
```

Spike设备扩展

以块设备为例

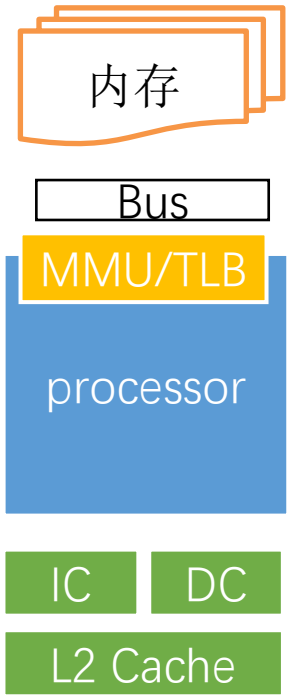
```
__attribute__((constructor)) static void on_load()
{
  static mmio_plugin_t test_mmio_plugin = {
      test_mmio_plugin_alloc,
      test_mmio_plugin_load,
      test_mmio_plugin_store,
      test_mmio_plugin_dealloc
  };

  register_mmio_plugin("test_mmio_plugin",
&test_mmio_plugin);
}
```

```
spike --extlib=<plugin-path>/plugin.so
        --device=test_mmio_plugin,0x10000000,<fs-path>
        --dtb=a.dtb
```

https://github.com/isrc-cas/PLCT-Open-Reports/blob/master/20200812-Linux设备树介绍及加载过程分析-王萌.pdf
https://www.bilibili.com/video/BV1Ti4y1g7oH

目录

Spike内存系统扩展



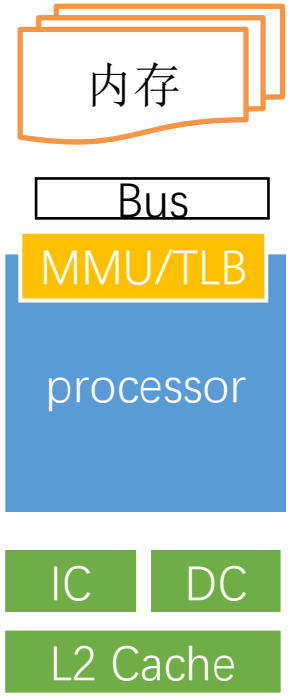
Spike内存系统扩展

取指

riscv/mmu.h

```
inline insn_fetch_t load_insn(reg_t addr)
{
  icache_entry_t entry;
  return refill_icache(addr, &entry)->data;
}
```

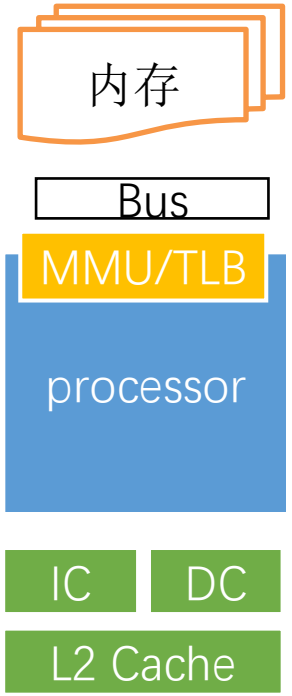
```
inline icache_entry_t* refill_icache(reg_t addr, icache_entry_t* entry)
{
  auto tlb_entry = translate_insn_addr(addr);
  insn_bits_t insn = from_le(*(uint16_t*)(tlb_entry.host_offset + addr));
  …
}
```



Spike内存系统扩展

取指

```
inline tlb_entry_t translate_insn_addr(reg_t addr) {
  reg_t vpn = addr >> PGSHIFT;
  if (likely(tlb_insn_tag[vpn % TLB_ENTRIES] == vpn))
    return tlb_data[vpn % TLB_ENTRIES];
  tlb_entry_t result;
  if (unlikely(tlb_insn_tag[vpn % TLB_ENTRIES] != (vpn | TLB_CHECK_TRIGGERS))) {
    result = fetch_slow_path(addr);
  } else {
    result = tlb_data[vpn % TLB_ENTRIES];
  }
  if (unlikely(tlb_insn_tag[vpn % TLB_ENTRIES] == (vpn | TLB_CHECK_TRIGGERS))) {
    uint16_t* ptr = (uint16_t*)(tlb_data[vpn % TLB_ENTRIES].host_offset + addr);
    int match = proc->trigger_match(OPERATION_EXECUTE, addr, from_le(*ptr));
    if (match >= 0) {
      throw trigger_matched_t(match, OPERATION_EXECUTE, addr, from_le(*ptr));
    }
  }
  return result;
}
```

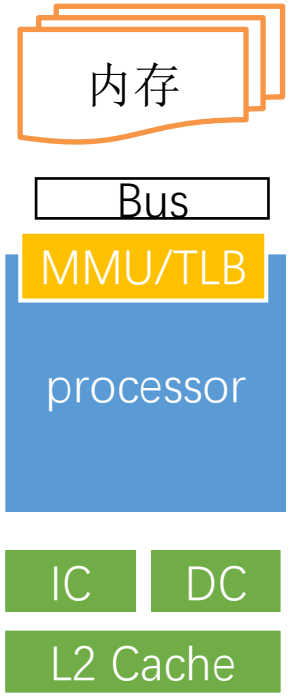


Spike内存系统扩展

取指

riscv/mmu.cc

```
tlb_entry_t mmu_t::fetch_slow_path(reg_t vaddr)
{
  reg_t paddr = translate(vaddr, sizeof(fetch_temp), FETCH, 0);

  if (auto host_addr = sim->addr_to_mem(paddr)) {
    return refill_tlb(vaddr, paddr, host_addr, FETCH);
  } else {
    if (!mmio_load(paddr, sizeof fetch_temp, (uint8_t*)&fetch_temp))
      throw trap_instruction_access_fault(vaddr, 0, 0);
    tlb_entry_t entry = {(char*)&fetch_temp - vaddr, paddr - vaddr};
    return entry;
  }
}
```

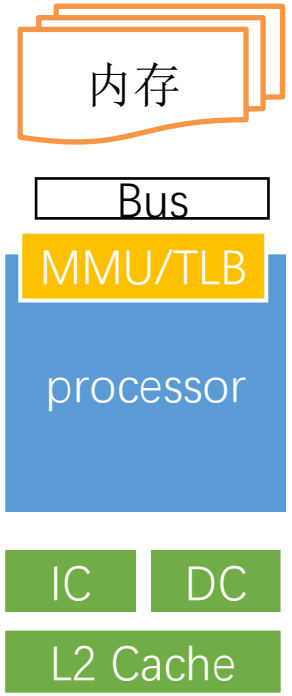


Spike内存系统扩展

取指

riscv/sim.cc

```
char* sim_t::addr_to_mem(reg_t addr) {
  if (!paddr_ok(addr))
    return NULL;
  auto desc = bus.find_device(addr);
  if (auto mem = dynamic_cast<mem_t*>(desc.second))
    if (addr - desc.first < mem->size())
      return mem->contents() + (addr - desc.first);
  return NULL;
}
```



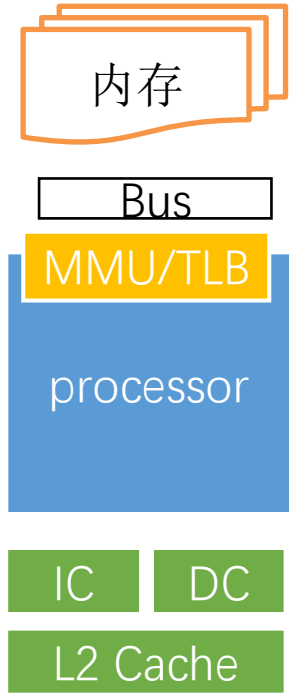
Spike内存系统扩展

加载内存

```
WRITE_RVC_RS2S(MMU.load_int32(RVC_RS1S + insn.rvc_lw_imm()));
```

```
#define load_func(type, prefix, xlate_flags) \
  inline type##_t prefix##_##type(reg_t addr) { \
...
    reg_t vpn = addr >> PGSHIFT; \
    size_t size = sizeof(type##_t); \
    if (likely(tlb_load_tag[vpn % TLB_ENTRIES] == vpn)) { \
      if (proc) READ_MEM(addr, size); \
      return from_target(*(target_endian<type##_t>*)(tlb_data[vpn %
TLB_ENTRIES].host_offset + addr)); \
    } \
  ...
    load_slow_path(addr, sizeof(type##_t), (uint8_t*)&res, (xlate_flags)); \
 ...
   return from_target(res)    }
```

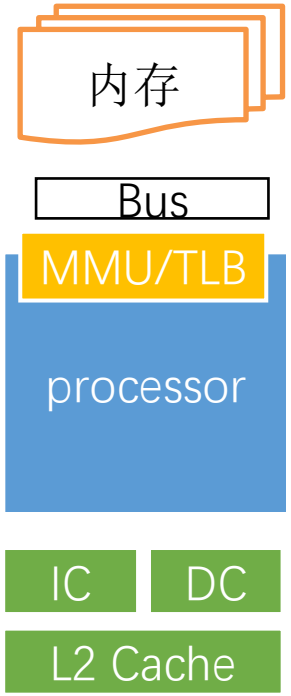
```
load_func(int8, load, 0)
load_func(int16, load, 0)
load_func(int32, load, 0)
load_func(int64, load, 0)
 ...
```



Spike内存系统扩展

加载内存

```
void mmu_t::load_slow_path(reg_t addr, reg_t len, uint8_t* bytes, uint32_t xlate_flags)
{
  reg_t paddr = translate(addr, len, LOAD, xlate_flags);

  if (auto host_addr = sim->addr_to_mem(paddr)) {
    memcpy(bytes, host_addr, len);
    if (tracer.interested_in_range(paddr, paddr + PGSIZE, LOAD))
      tracer.trace(paddr, len, LOAD);
    else
      refill_tlb(addr, paddr, host_addr, LOAD);
  } else if (!mmio_load(paddr, len, bytes)) {
    throw trap_load_access_fault(addr, 0, 0);
  }
  ...
}
```



Spike内存系统扩展

以snapshot功能为例: 为了降低存储空间，snapshot采用类似于copy-on-write的策略，将对修改的内存进行保存

```
void mmu_t::store_slow_path(reg_t addr, reg_t len, const uint8_t* bytes,
uint32_t xlate_flags)
{
  reg_t paddr = translate(addr, len, STORE, xlate_flags);
...
  if (auto host_addr = sim->addr_to_mem(paddr)) {
    (*sim -> get_tags())[paddr >> PGSHIFT] = true;
    memcpy(host_addr, bytes, len);
    if (tracer.interested_in_range(paddr, paddr + PGSIZE, STORE))
      tracer.trace(paddr, len, STORE);
    else
      refill_tlb(addr, paddr, host_addr, STORE);
  } else if (!mmio_store(paddr, len, bytes)) {
    throw trap_store_access_fault(addr, 0, 0);
  }
}
```

未来工作

- 继续完成Zfinx扩展支持
- 新的指令集扩展支持
- 进一步完善块设备的支持，并尝试添加新的设备支持
- 欢迎伙伴们提出自己的需求

  https://github.com/isrc-cas/plct-spike

- **欢迎实习生加入!**

谢谢各位

欢迎提问、讨论、交流合作