

9.支持RV32IM的NEMU (PA 2.1)

RTFSC

ISA相关API说明文档

全局类型

- `word_t` 与ISA字长等长的无符号类型
- `sword_t` 与ISA字长等长的有符号类型
- `char *FMT_WORD` 与 `word_t` 类型对应的十六进制格式化说明符, 在32位的ISA中为 `"0x%08x"`, 在64位的ISA中为 `"0x%016lx"`.

Monitor相关

- `unsigned char isa_logo[]` 用于在未实现指令的报错信息中提示开发者阅读相关的手册.
- `word_t RESET_VECTOR` 表示PC的初始值
- `void init_isa()` 在monitor初始化时调用, 进行至少如下ISA相关的初始化工作:
 - 设置必要的寄存器初值, 如 `PC` 等
 - 加载内置客户程序

寄存器相关

- `struct { word_t pc } CPU_state` 寄存器结构的类型定义, 其中必须包含一个名为 `pc`, 类型为 `word_t` 的成员.
- `CPU_state cpu` 寄存器结构的全局定义.
- `void isa_reg_display()` 用于打印寄存器当前的值.
- `word_t isa_reg_str2val(const char *name, bool *success);` 若存在名称为 `name` 的寄存器, 则返回其当前值, 并设置 `success` 为 `true`; 否则设置 `success` 为 `false`.

指令执行相关

- `struct { } ISADecodeInfo` 用于存放ISA相关的译码信息, 会嵌入在译码信息结构体 `Decode` 的定义中.
- `int isa_exec_once(Decode *s)` 取出 `s->pc` 指向的指令并译码执行, 同时更新 `s->snpc`.

虚拟内存相关

- `int isa_mmu_check(vaddr_t vaddr, int len, int type);` 检查当前系统状态下对内存区间为 `[vaddr, vaddr + len)`, 类型为 `type` 的访问是否需要经过地址转换.

- `paddr_t isa_mmu_translate(vaddr_t vaddr, int len, int type);` 对内存区间为 `[vaddr, vaddr + len)`, 类型为 `type` 的内存访问进行地址转换. 函数返回值可能为:

中断异常相关

- `vaddr_t isa_raise_intr(word_t NO, vaddr_t epc);` 抛出一个号码为 `NO` 的异常, 其中 `epc` 为触发异常的指令PC, 返回异常处理的出口地址.
- `word_t isa_query_intr()` 查询当前是否有未处理的中断, 若有则返回中断号码, 否则返回 `INTR_EMPTY`.

DiffTest相关

- `bool isa_difftest_checkregs(CPU_state *ref_r, vaddr_t pc);` 检查当前的寄存器状态是否与 `ref_r` 相同, 其中 `pc` 为 `cpu.pc` 的上一条动态指令的PC, 即 `cpu.pc` 的旧值. 如果状态相同, 则返回 `true`, 否则返回 `false`.
- `void isa_difftest_attach()` 将当前的所有状态同步到REF, 并在之后的执行中开启 DiffTest.

指令执行过程

整体调用顺序

按下 `c` 之后会调用 `cpu_exec(-1)`

`cpu_exec(n) -> execute(n) -> n exec_once(Decode *s, vaddr_t pc)`

`exec_once` 代码如下:

```
static void exec_once(Decode *s, vaddr_t pc) {
    s->pc = pc;
    s->snpc = pc;
    isa_exec_once(s);
    cpu.pc = s->dnpc;
#ifdef CONFIG_ITRACE
    // ... Trace 相关代码
}
```

这个函数的功能就是, 让CPU执行当前PC指向的一条指令, 然后更新PC。

- `int isa_exec_once(Decode *s)` 取出 `s->pc` 指向的指令并译码执行, 同时更新 `s->snpc`.

`exec_once` 接收一个 `Decode` 类型的结构体指针 `s`, 这个结构体用于存放一条指令执行过程中所需要的信息 (指令PC、下一条指令的PC等, 还有ISA相关信息在 `nemu/src/isa/$ISA/include/isa-def.h` 中)

```
typedef struct Decode {
    vaddr_t pc;
    vaddr_t snpc; // static next pc
    vaddr_t dnpc; // dynamic next pc
    ISADecodeInfo isa;
    IFDEF(CONFIG_ITRACE, char logbuf[128]);
} Decode;
```

- `pc` 是当前指令
- `snpc` 指下一条静态指令（程序代码中的指令）
- `dnpc` 指下一条动态指令（运行过程中的指令，例如出现跳转指令，就会指向跳转地址）
- `ISADecodeInfo` ISA相关信息 - RV32中是 `union {uint32_t val;} inst`

事实上 `exec_once()` 函数覆盖了指令周期的所有阶段：取值、译码、执行、更新PC

取值 (instruction fetch , IF)

`isa_exec_once()` 做的第一件事情就是取指令，== `isa_exec_once` 代码如下

```
int isa_exec_once(Decode *s) {
    s->isa.inst.val = inst_fetch(&s->snpc, 4);
    return decode_exec(s);
}
```

`inst_fetch` 专门负责取指令工作，根据 `s->snpc` 中的地址去取出指令，并且根据 `len = 4`，来更新 `snpc`，取出的指令被放置在 `s->isa.inst.val` 中。

最后交由 `decode_exec` 执行

`inst_fetch` 代码如下

```
static inline uint32_t inst_fetch(vaddr_t *pc, int len) {
    uint32_t inst = vaddr_ifetch(*pc, len);
    (*pc) += len;
    return inst;
}
```

`vaddr_ifetch()` 代码如下

```
word_t vaddr_ifetch(vaddr_t addr, int len) {
    return paddr_read(addr, len);
}
```

```
}
```

实际上就是一次读操作

译码(instruction decode, ID)

接下来 `isa_exec_once` 会进入 `decode_exec()` 函数, `decode_exec` 代码如下

```
static int decode_exec(Decode *s) {
    int rd = 0;
    word_t src1 = 0, src2 = 0, imm = 0;
    s->dnpc = s->snpc;

#define INSTPAT_INST(s) ((s)->isa.inst.val)
#define INSTPAT_MATCH(s, name, type, ... /* execute body */ ) { \
    decode_operand(s, &rd, &src1, &src2, &imm, concat(TYPE_, type)); \
    __VA_ARGS__ ; \
}

    INSTPAT_START();
    INSTPAT("??????? ???? ???? ??? ????? 00101 11", auipc , U, R(rd) = s->pc
    INSTPAT("??????? ???? ???? 100 ????? 00000 11", lbu , I, R(rd) = Mr(sr
    // ...
    INSTPAT_END();

    R(0) = 0; // reset $zero to 0

    return 0;
}
```

`INSTPAT` 格式如下

`INSTPAT` (模式字符串, 指令名称, 指令类型, 指令执行操作)

相当于依次遍历这几个模式, 如果匹配, 则执行指令, 然后跳转到结尾。

- `INSTPAT_INST(s)` 指定如何去取 `Decode s` 中的指令值
- `INSTPAT_MATCH` 指定了如何去解析指令中的值, 使用 `decode_operand`

以下是 `decode_operand` 的代码

```
static void decode_operand(Decode *s, int *rd, word_t *src1, word_t *src2, wo
    uint32_t i = s->isa.inst.val;
```

```

int rs1 = BITS(i, 19, 15);
int rs2 = BITS(i, 24, 20);
*rd     = BITS(i, 11, 7);
switch (type) {
    case TYPE_I: src1R();          immI(); break;
    case TYPE_U:          immU(); break;
    case TYPE_S: src1R(); src2R(); immS(); break;
}
}

```

执行(execute, EX)

执行模式匹配之后的操作

更新PC

将 `s->dnpc` 赋值给 `cpu.pc`，dnpc是动态维护的

运行第一个C程序

在NEMU中实现上文提到的指令，具体细节请务必参考手册。实现成功后，在NEMU中运行客户程序 `dummy`，你将会看到 `HIT GOOD TRAP` 的信息。如果你没有看到这一信息，说明你的指令实现不正确，你可以使用PA1中实现的简易调试器帮助你调试。

```

ary, you can disable it in menuconfig
[src/monitor/monitor.c:32 welcome] Build time: 20:29:45, Nov 27 2023
Welcome to riscv32-NEMU!
For help, type "help"
(nemu) c
[src/cpu/cpu-exec.c:129 cpu_exec] nemu: HIT GOOD TRAP at pc = 0x80000030
[src/cpu/cpu-exec.c:97 statistic] host time spent = 112 us
[src/cpu/cpu-exec.c:98 statistic] total guest instructions = 13
[src/cpu/cpu-exec.c:99 statistic] simulation frequency = 116,071 inst/s
(nemu) q
dummy
[ dummy] PASS!
miical@Miicals-BOSC-Ubuntu ~/ysyx-workbench/am-kernels/tests/cpu-tests$

```

✅ 完成运行第一个C程序

添加四个指令

```

INSTPAT("???????? ???? ???? 000 ????? 00100 11", addi    , I, R(rd) = src1
INSTPAT("???????? ???? ???? 000 ????? 11001 11", jalr    , I, R(rd) = s->pc

```

```
INSTPAT("??????? ???? ???? 010 ????? 01000 11", sw      , S, Mw(src1 + imm
INSTPAT("??????? ???? ???? ??? ????? 11011 11", jal     , J, R(rd) = s->pc
```

注意添加立即数时的格式

```
#define immJ() do { *imm = ((SEXT(BITS(i, 31, 31), 1) << 19 | BITS(i, 19, 12)
```




运行更多程序

已支持指令：

- **R型**
 - add
 - sub
 - sltu
 - snez
 - xor
 - or
 - sll
 - mul
 - slt
 - rem
 - mulh
 - remu
 - divu
 - div
 - sra
 - srl
- **I型**
 - addi
 - li
 - mv
 - nop
 - lbu
 - jalr
 - ret
 - lw
 - sltiu

- seqz
- srai
- andi
- xori
- lh
- lhu
- slli
- srli
- **S型**
 - sb
 - sw
 - sh
- **B型**
 - beq
 - beqz
 - bne
 - bnez
 - bge
 - blez
 - blt
 - bltu
 - bgeu
- **U型**
 - auipc
 - lui
- **J型**
 - jal
 - j
- **N**
 - ebreak
 - inv

测试点通过顺序 指令添加数:

-  sum 6
-  fib 1
-  add 0

- ☒ add-longlong 3
- ☒ bit 6
- ☒ bubble-sort 1
- ☒ fact 1
- ☒ if-else 2
- ☒ leap-year 1
- ☒ load-store 4
- ☒ matrix-mul 0
- ☒ max 0
- ☒ mersenne 3
- ☒ min3 0
- ☒ mov-c 0
- ☒ movsx 0
- ☒ mul-longlong 0
- ☒ pascal 0
- ☒ prime 0
- ☒ quick-sort 0
- ☒ recursion 1
- ☒ select-sort 0
- ☒ shift 0
- ☒ shuixianhua 0
- ☒ sub-longlong 0
- ☒ switch 1
- ☒ to-lower-case 0
- ☒ unalign 0
- ☒ wanshu 0
- ☒ crc32 1
- ☒ goldbach 0