从零开始的RISC-V模拟器开发

第7讲 QEMU篇之CPU虚拟化1

中国科学院软件研究所
PLCT实验室

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

本课内容

CPU虚拟化

- RISCV CPU实现分析
- 新RISCV CPU建立
- TCG原理与指令翻译
- ~~DecodeTree与NICE指令添加~~
- ~~CSR寄存器实现添加~~

QEMU中的CPU Model

QEMU中RISC-V CPU的支持

```
$qemu-system-riscv32 -cpu ?
any
lowrisc-ibex
rv32
sifive-e31
sifive-e34
sifive-u34
```

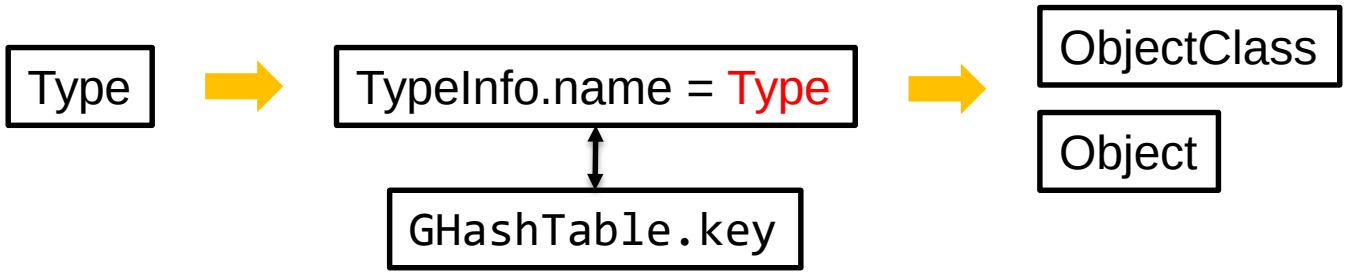
```
$qemu-system-riscv64 -cpu ?
any
rv64
sifive-e51
sifive-u54
```

QOM之TYPE定义

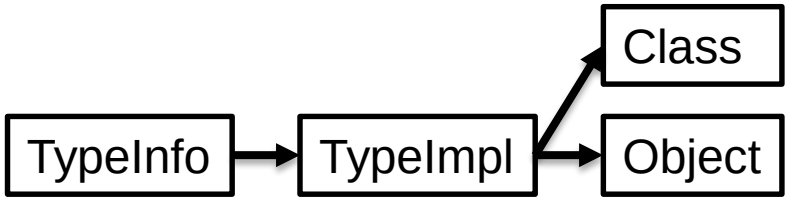
文件：target/riscv/cpu.h

registering user creatable types

```
#define TYPE_RISCV_CPU "riscv-cpu"

#define RISCV_CPU_TYPE_SUFFIX "-" TYPE_RISCV_CPU
#define RISCV_CPU_TYPE_NAME(name) (name RISCV_CPU_TYPE_SUFFIX)
#define CPU_RESOLVING_TYPE TYPE_RISCV_CPU

#define TYPE_RISCV_CPU_ANY              RISCV_CPU_TYPE_NAME("any")
#define TYPE_RISCV_CPU_BASE32           RISCV_CPU_TYPE_NAME("rv32")
#define TYPE_RISCV_CPU_BASE64           RISCV_CPU_TYPE_NAME("rv64")
#define TYPE_RISCV_CPU_IBEX             RISCV_CPU_TYPE_NAME("lowrisc-ibex")
#define TYPE_RISCV_CPU_SIFIVE_E31       RISCV_CPU_TYPE_NAME("sifive-e31")
#define TYPE_RISCV_CPU_SIFIVE_E34       RISCV_CPU_TYPE_NAME("sifive-e34")
#define TYPE_RISCV_CPU_SIFIVE_E51       RISCV_CPU_TYPE_NAME("sifive-e51")
#define TYPE_RISCV_CPU_SIFIVE_U34       RISCV_CPU_TYPE_NAME("sifive-u34")
#define TYPE_RISCV_CPU_SIFIVE_U54       RISCV_CPU_TYPE_NAME("sifive-u54")
```



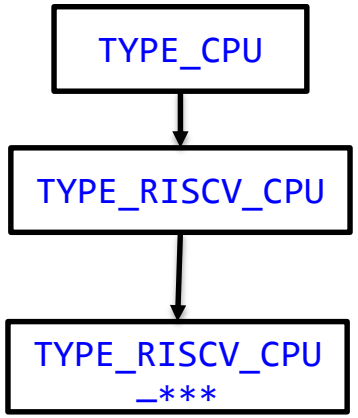
RISC-V CPU的Class与Object



Class:

```
struct RISCVCPUClass {
    /*< private >*/
    CPUClass parent_class;
    /*< public >*/
    DeviceRealize parent_realize;
    DeviceReset parent_reset;
};
```

部分TYPE线



实例Object:

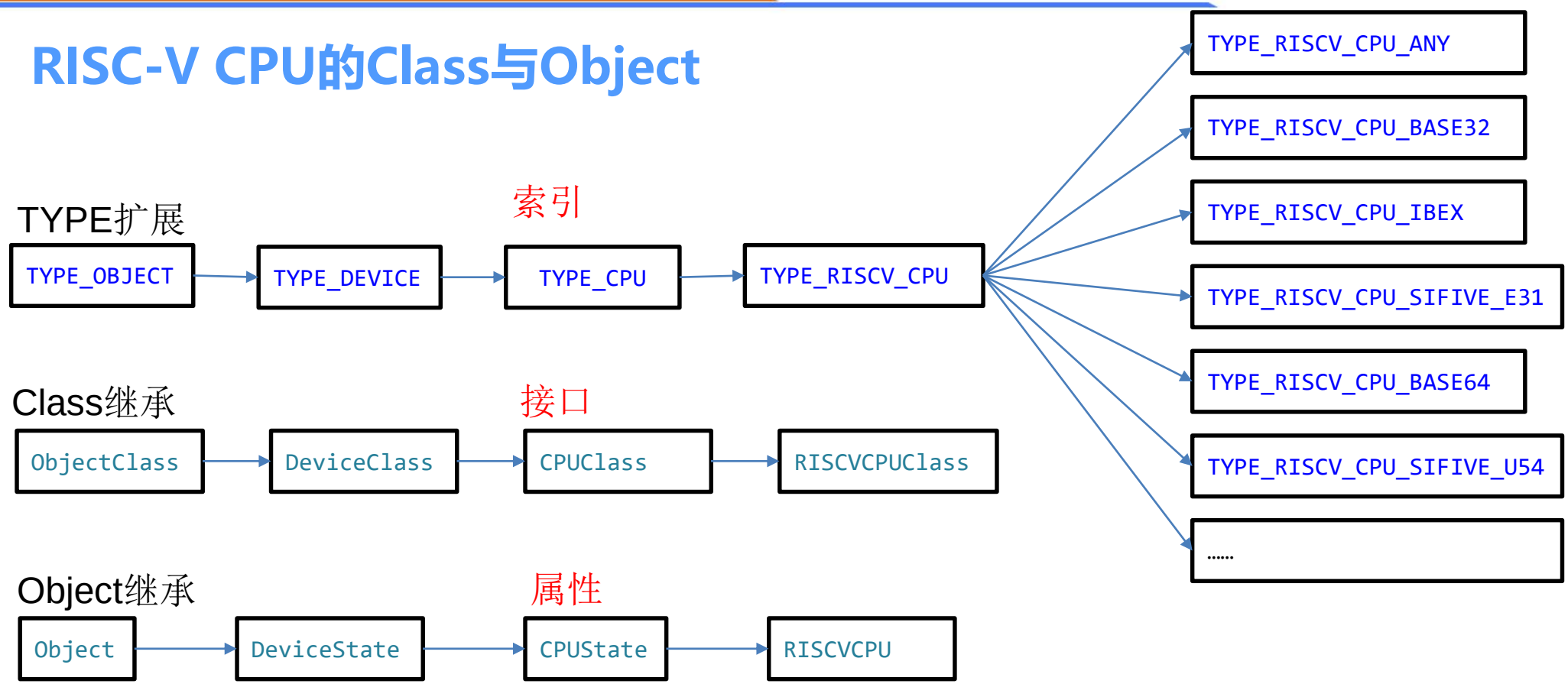
```
struct RISCVCPU {
    /*< private >*/
    CPUState parent_obj;
    /*< public >*/
    CPUNegativeOffsetState neg;
    CPURISCVState env;

    char *dyn_csr_xml;

    /* Configuration Settings */
    struct {
        ……
          } cfg;
};
```

```
#define DEFINE_CPU(type_name, initfn)      \
    {                                      \
        .name = type_name,                 \
        .parent = TYPE_RISCV_CPU,          \
        .instance_init = initfn            \
    }
```

文件:target/riscv/cpu.c

```
static const TypeInfo riscv_cpu_type_infos[] = {
    {
        .name = TYPE_RISCV_CPU,
        .parent = TYPE_CPU,
        .instance_size = sizeof(RISCVCPU),
        .instance_align = __alignof__(RISCVCPU),
        .instance_init = riscv_cpu_init,
        .abstract = true,
        .class_size = sizeof(RISCVCPUClass),
        .class_init = riscv_cpu_class_init,
    },
    DEFINE_CPU(TYPE_RISCV_CPU_ANY,              riscv_any_cpu_init),
#if defined(TARGET_RISCV32)
    DEFINE_CPU(TYPE_RISCV_CPU_BASE32,           rv32_base_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_IBEX,             rv32_ibex_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_SIFIVE_E31,       rv32_sifive_e_cpu_init),
    ……
#elif defined(TARGET_RISCV64)
    DEFINE_CPU(TYPE_RISCV_CPU_BASE64,           rv64_base_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_SIFIVE_E51,       rv64_sifive_e_cpu_init),
    DEFINE_CPU(TYPE_RISCV_CPU_SIFIVE_U54,       rv64_sifive_u_cpu_init),
#endif
};

DEFINE_TYPES(riscv_cpu_type_infos)
```

RISC-V CPU的Class与Object



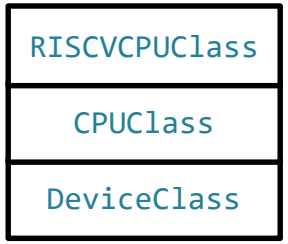
RISC-V CPU的Class

Class:

```
struct RISCVCPUClass {
    /*< private >*/
    CPUClass parent_class;
    /*< public >*/
    DeviceRealize parent_realize;
    DeviceReset parent_reset;
};
```

```
struct DeviceClass {
    /*< private >*/
    ObjectClass parent_class;
    /*< public >*/

    DECLARE_BITMAP(categories, DEVICE_CATEGORY_MAX);
    const char *fw_name;
    const char *desc;

    Property *props_;

    bool user_creatable;
    bool hotpluggable;

    DeviceReset reset;
    DeviceRealize realize;
    DeviceUnrealize unrealize;
    ……
};
```

Class继承




```
struct CPUClass {
    DeviceClass parent_class;
    ObjectClass *(*class_by_name)();
    void (*parse_features)();
    int reset_dump_flags;
    bool (*has_work)();
    bool (*virtio_is_big_endian)();
    int (*memory_rw_debug)();
    void (*dump_state)();
    GuestPanicInformation* (*get_crash_info)();
    void (*dump_statistics)();
    int64_t (*get_arch_id)();
    bool (*get_paging_enabled)();
    void (*get_memory_mapping)();
    void (*set_pc)();
    hwaddr (*get_phys_page_debug)();
    hwaddr (*get_phys_page_attrs_debug)();
    int (*asidx_from_attrs)();
    int (*gdb_read_register)();
    int (*gdb_write_register)();
    int (*write_elf64_note)();
    int (*write_elf64_qemunote)();
    int (*write_elf32_note)();
    int (*write_elf32_qemunote)();
    const VMStateDescription *vmsd;
    const char *gdb_core_xml_file;
    gchar * (*gdb_arch_name)();
    const char * (*gdb_get_dynamic_xml)();
    void (*disas_set_info)();
    const char *deprecation_note;
    int gdb_num_core_regs;
    bool gdb_stop_before_watchpoint;
    struct AccelCPUClass *accel_cpu;
    struct TCGCPUOps *tcg_ops;
};
```

RISC-V CPU的Class

Class:

```
struct RISCVCPUClass {
    /*< private >*/
    CPUClass parent_class;
    /*< public >*/
    DeviceRealize parent_realize;
    DeviceReset parent_reset;
};
```

```
struct CPUClass {
    struct TCGCPUOps *tcg_ops;
};
```

文件:include/hw/core/tcg-cpu-ops.h

```
struct TCGCPUOps {
    void (*initialize)(void);
    void (*synchronize_from_tb)(CPUState *cpu, const TranslationBlock *tb);
    void (*cpu_exec_enter)(CPUState *cpu);
    void (*cpu_exec_exit)(CPUState *cpu);
    bool (*cpu_exec_interrupt)(CPUState *cpu, int interrupt_request);
    void (*do_interrupt)(CPUState *cpu);
    bool (*tlb_fill)(CPUState *cpu, vaddr address, int size,
                     MMUAccessType access_type, int mmu_idx,
                     bool probe, uintptr_t retaddr);
    void (*debug_excp_handler)(CPUState *cpu);
#ifdef NEED_CPU_H
#ifdef CONFIG_SOFTMMU
    void (*do_transaction_failed)(CPUState *cpu, hwaddr physaddr, vaddr addr,
                                  unsigned size, MMUAccessType access_type,
                                  int mmu_idx, MemTxAttrs attrs,
                                  MemTxResult response, uintptr_t retaddr);
    void (*do_unaligned_access)(CPUState *cpu, vaddr addr,
                                MMUAccessType access_type,
                                int mmu_idx, uintptr_t retaddr);
    vaddr (*adjust_watchpoint_address)(CPUState *cpu, vaddr addr, int len);
    bool (*debug_check_watchpoint)(CPUState *cpu, CPUWatchpoint *wp);
    bool (*io_recompile_replay_branch)(CPUState *cpu,
                                       const TranslationBlock *tb);
#endif /* CONFIG_SOFTMMU */
#endif /* NEED_CPU_H */
};
```

```
static struct TCGCPUOps riscv_tcg_ops = {
    .initialize = riscv_translate_init,
    .synchronize_from_tb = riscv_cpu_synchronize_from_tb,
    .cpu_exec_interrupt = riscv_cpu_exec_interrupt,
    .tlb_fill = riscv_cpu_tlb_fill,

#ifndef CONFIG_USER_ONLY
    .do_interrupt = riscv_cpu_do_interrupt,
    .do_transaction_failed = riscv_cpu_do_transaction_failed,
    .do_unaligned_access = riscv_cpu_do_unaligned_access,
#endif /* !CONFIG_USER_ONLY */
};
```

设置函数:

```
device_class_set_parent_realize(dc, riscv_cpu_realize,&mcc->parent_realize);
device_class_set_parent_reset(dc, riscv_cpu_reset, &mcc->parent_reset);
```

RISC-V CPU的Object

实例Object:

```
struct RISCVCPU {
    /*< private >*/
    CPUState parent_obj;
    /*< public >*/
    CPUNegativeOffsetState neg;
    CPURISCVState env;

    char *dyn_csr_xml;

    /* Configuration Settings */
    struct {
        ……
          } cfg;
};
```

```
struct {
        bool ext_i;
        bool ext_e;
        bool ext_g;
        bool ext_m;
        bool ext_a;
        bool ext_f;
        bool ext_d;
        bool ext_c;
        bool ext_s;
        bool ext_u;
        bool ext_h;
        bool ext_v;
        bool ext_counters;
        bool ext_ifencei;
        bool ext_icsr;

        char *priv_spec;
        char *user_spec;
        char *vext_spec;
        uint16_t vlen;
        uint16_t elen;
        bool mmu;
        bool pmp;
    } cfg;
```

RISC-V寄存器(参考spec):
X0~X31 for int XLEN
F0~F31 for fd FLEN
V0~V32 for vector VLEN/SEW
CSR(Control and Status Register)寄存器

文件：target/riscv/cpu.h

```
struct CPURISCVState {
    target_ulong gpr[32];
    uint64_t fpr[32]; /* assume both F and D extensions */

    /* vector coprocessor state. */
    uint64_t vreg[32 * RV_VLEN_MAX / 64] QEMU_ALIGNED(16);
    //vector reg
    target_ulong pc;
    target_ulong misa;

    uint32_t features;
    /* Hypervisor CSRs */
    /* Virtual CSRs */
    /* HS Backup CSRs */
    /* temporary htif regs */
    /* physical memory protection */
    /* machine specific rdtime callback */
    /* True if in debugger mode.  */
    bool debugger;

    float_status fp_status;
    /* Fields from here on are preserved across CPU reset. */
    QEMUTimer *timer; /* Internal timer */
};
```

RISC-V CPU的实例化函数

RISCV CPU TypeInfo注册：

```
.instance_init = riscv_cpu_init,
.class_init = riscv_cpu_class_init,
```

Class

特定RISCV CPU TypeInfo注册(以E31为例)：

```
.instance_init =  rvxx_sifive_e_cpu_init,
.class_init = riscv_cpu_class_init,
```

Object

注册定义宏：

```
DEFINE_CPU(TYPE_RISCV_CPU_SIFIVE_E31,
           rvxx_sifive_e_cpu_init),
```

```
static void rvxx_sifive_e_cpu_init(Object *obj)
{
    CPURISCVState *env = &RISCV_CPU(obj)->env;
    set_misa(env, RVXLEN | RVI | RVM | RVA | RVC | RVU);
    set_priv_version(env, PRIV_VERSION_1_10_0);
    set_resetvec(env, 0x1004);
    qdev_prop_set_bit(DEVICE(obj), "mmu", false);
}
```

文件:target/riscv/cpu.c

```
static void riscv_cpu_class_init(ObjectClass *c, void *data)
```

- parent_realize与parent_reset
- GDB相关接口
- cpu索引，状态判断，PC设置功能接口(基于env)
- tcg_ops操作接口
- riscv_cpu_properties属性接口

```
static Property
riscv_cpu_properties[]
```

GHashTable

env设置接口:

```
void set_misa(CPURISCVState *env, target_ulong misa)
void set_priv_version(CPURISCVState *env, int priv_ver)
void set_vext_version(CPURISCVState *env, int vext_ver)
void set_feature(CPURISCVState *env, int feature)
void set_resetvec(CPURISCVState *env, int resetvec)
```

作用于

```
struct RISCVCPU
 CPURISCVState env
```

- misa
- priv_ver
- vext_ver
- features
- resetvec

属性设置接口:

```
void qdev_prop_set_bit(DeviceState *dev,……)
……
void qdev_prop_set_uint32(DeviceState *dev,……)
```

修改Property
更新env
riscv_cpu_realize使用

NUCLEI CPU的添加

以N600为示例：

```
#define TYPE_RISCV_CPU_NUCLEI_N600    RISCV_CPU_TYPE_NAME("nuclei-n600")

static const TypeInfo riscv_cpu_type_infos[] = {
  ……
  DEFINE_CPU(TYPE_RISCV_CPU_NUCLEI_N600,    rv32_nuclei_n_cpu_init),
  ……
};

static void rv32_nuclei_n_cpu_init(Object *obj)
{
    CPURISCVState *env = &RISCV_CPU(obj)->env;
    set_misa(env, RV32 | RVI | RVM | RVA | RVC | RVF | RVD | RVU);
    set_priv_version(env, PRIV_VERSION_1_10_0);
    qdev_prop_set_bit(DEVICE(obj), "mmu", false);
    set_resetvec(env, DEFAULT_RSTVEC);
    set_feature(env, RISCV_FEATURE_PMP);
}
```

qemu命令示例(开关指令扩展):
qemu-system-riscv32 -cpu nuclei-n600,x-nice=true
qemu-system-riscv32 -cpu rv32,f=true,d=true, x-v=false

CORE系列	RISC-V支持状态
N600	RV32I/M/A/C/F/D/P
NX600	RV64I/M/A/C/F/D/P
UX600	RV64I/M/A/C/F/D/P

运行结果:

$qemu-system-riscv32 -cpu ?	$qemu-system-riscv64 -cpu ?
any	any
lowrisc-ibex	nuclei-nx600
nuclei-n600	nuclei-ux600
rv32	rv64
sifive-e31	sifive-e51
sifive-e34	sifive-u54
sifive-u34	

定义于:target/riscv/cpu.c

```
static Property riscv_cpu_properties[]
```

作用于: `riscv_cpu_realize()`

```
static void rv32_base_cpu_init(Object *obj)
{
    CPURISCVState *env = &RISCV_CPU(obj)->env;
    /* We set this in the realise function */
    set_misa(env, RV32);
}
```

NUCLEI CPU的添加

Machine中NucLeiNSoCState扩充 ps:原NucleiHBSoCState

```
typedef struct NucLeiNSoCState
{
    /*< private >*/
    SysBusDevice parent_obj;

    /*< public >*/
}  NucLeiNSoCState;
```

```
typedef struct  NucLeiNSoCState
{
    /*< private >*/
    SysBusDevice parent_obj;

    /*< public >*/
    RISCVHartArrayState cpus;
    //RISCVCPU cpu;

}  NucLeiNSoCState;
```

关于Hart？硬件线程

socket？ 槽数目

```
struct RISCVHartArrayState {
    /*< private >*/
    SysBusDevice parent_obj;

    /*< public >*/
    uint32_t num_harts;
    uint32_t hartid_base;
    char *cpu_type;
    uint64_t resetvec;
    RISCVCPU *harts;
};
```

TYPE_SYS_BUS_DEVICE

```
static Property riscv_harts_props[]
num-harts
hartid-base
cpu-type
resetvec
```

文件:hw/riscv/riscv_hart.c

```
static void riscv_harts_realize(DeviceState *dev, Error **errp)
{
    RISCVHartArrayState *s = RISCV_HART_ARRAY(dev);
    int n;

    s->harts = g_new0(RISCVCPU, s->num_harts);

    for (n = 0; n < s->num_harts; n++) {
        if (!riscv_hart_realize(s, n, s->cpu_type, errp)) {
            return;
        }
    }
}
```

qdev_realize

```
static void riscv_harts_class_init(ObjectClass *klass, void *data)
{
    DeviceClass *dc = DEVICE_CLASS(klass);

    device_class_set_props(dc, riscv_harts_props);
    dc->realize = riscv_harts_realize;
}
```

```
static const TypeInfo riscv_harts_info = {
    .name          = TYPE_RISCV_HART_ARRAY,
    .parent        = TYPE_SYS_BUS_DEVICE,
    .instance_size = sizeof(RISCVHartArrayState),
    .class_init    = riscv_harts_class_init,
};
```

NUCLEI CPU的添加

Machine启动打印流程：

```
>>nuclei_mcu_machine_class_init
>>nuclei_n_soc_class_init
>>nuclei_machine_instance_init
>>nuclei_mcu_machine_init (原nuclei_board_init)
>>nuclei_n_soc_instance_init (原nuclei_soc_init)
>>nuclei_n_soc_realize
```

```
struct MachineClass {
    ……
    void (*init)(MachineState *state);
    void (*smp_parse)(MachineState *ms, QemuOpts *opts);
    int max_cpus;
    int min_cpus;
    int default_cpus;
    const char *default_cpu_type;
    bool has_hotpluggable_cpus;
    const char **valid_cpu_types;
    const char *default_ram_id;
    CpuInstanceProperties (*cpu_index_to_instance_props)();
    const CPUArchIdList *(*possible_cpu_arch_ids)(MachineState *machine);
    int64_t (*get_default_cpu_node_id)(const MachineState *ms, int idx);
};
```

nuclei_mcu_machine_class_init

```
mc->max_cpus = 1;
mc->default_cpu_type = NUCLEI_N_CPU;
```

update MachineClass
设置mc->init

update DeviceClass
设置dc->realize

nuclei_n_soc_class_init

update MachineState

nuclei_machine_instance_init

mc->init

nuclei_mcu_machine_init

```
object_initialize_child(OBJECT(machine), "soc",
                        &s->soc, TYPE_NUCLEI_HBIRD_SOC);
qdev_realize(DEVICE(&s->soc), NULL, &error_abort);
```

nuclei_n_soc_instance_init

```
object_initialize_child(obj, "cpus", &s->cpus, TYPE_RISCV_HART_ARRAY);
object_property_set_int(OBJECT(&s->cpus), "num-harts", ms->smp.cpus,
                        &error_abort);
```

dc->realize

nuclei_n_soc_realize

```
object_property_set_str(OBJECT(&s->cpus), "cpu-type", ms->cpu_type,
                        &error_abort);
sysbus_realize(SYS_BUS_DEVICE(&s->cpus), &error_abort);
```

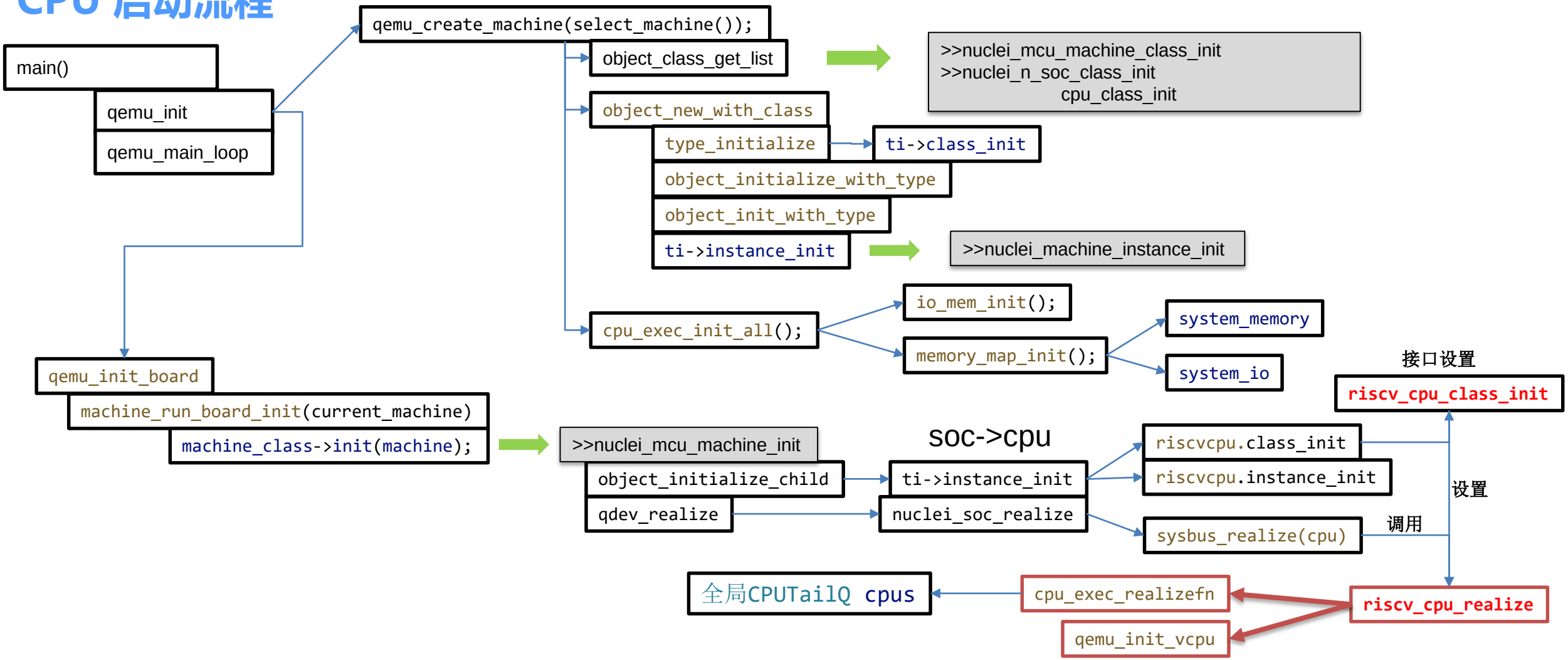
NUCLEI CPU的添加

```
$qemu-system-riscv32 -nographic -M mcu_200t  -cpu nuclei-n600
>>nuclei_mcu_machine_class_init
>>nuclei_n_soc_class_init
>>nuclei_machine_instance_init
>>nuclei_mcu_machine_init
>>nuclei_n_soc_instance_init
>>nuclei_n_soc_realize
QEMU 5.2.90 monitor - type 'help' for more information
(qemu) info qom-tree
/machine (mcu_200t-machine)
  /peripheral (container)
  /peripheral-anon (container)
  /soc (riscv.nuclei.n.soc)
    /cpus (riscv.hart_array)
      /harts[0] (nuclei-n600-riscv-cpu)
  /unattached (container)
    /io[0] (memory-region)
    /sysbus (System)
    /system[0] (memory-region)
```

```
$qemu-system-riscv64 -nographic -M ddr_200t -cpu nuclei-ux600
>>nuclei_mcu_machine_class_init
>>nuclei_n_soc_class_init
QEMU 5.2.90 monitor - type 'help' for more information
(qemu) info qom-tree
/machine (ddr_200t-machine)
  /peripheral (container)
  /peripheral-anon (container)
  /soc (riscv.nuclei.u.soc)
    /cpus (riscv.hart_array)
      /harts[0] (nuclei-ux600-riscv-cpu)
  /unattached (container)
    /io[0] (memory-region)
    /sysbus (System)
    /system[0] (memory-region)
```

CPU 工作原理

CPU 启动流程



CPU 启动流程

qemu_init_vcpu → cpus_accel->create_vcpu_thread(cpu);

加速器设置：

```
configure_accelerators(argv[0]);
```

-accel select accelerator (kvm, xen, hax, hvf, whpx or tcg)

文件：accel/tcg/tcg-accel-ops.c

```
static void tcg_accel_ops_init(AccelOpsClass *ops)
{
    if (qemu_tcg_mttcg_enabled()) {
        ops->create_vcpu_thread = mttcg_start_vcpu_thread;
        ……
    } else if (icount_enabled()) {
        ops->create_vcpu_thread = rr_start_vcpu_thread;
        ops->kick_vcpu_thread = rr_kick_vcpu_thread;
        ……
    } else {
        ops->create_vcpu_thread = rr_start_vcpu_thread;
        ……
    }
}
```

```
qemu_thread_create(cpu->thread, thread_name,
                   mttcg_cpu_thread_fn/rr_cpu_thread_fn,
                    cpu, QEMU_THREAD_JOINABLE);
```

while or do ... while()运行以下:

```
if (cpu_can_run(cpu)) {
            int r;
            ……
            r = tcg_cpus_exec(cpu);
            ……
}
```

核心代码片段

```
int tcg_cpus_exec(CPUState *cpu)
{
    int ret;
    ……
    cpu_exec_start(cpu);
    ret = cpu_exec(cpu);
    cpu_exec_end(cpu);
    ……
    return ret;
}
```

CPU 启动流程

文件:accel/tcg/cpu-exec.c

cc->tcg_ops->do_interrupt(cpu)

cc->tcg_ops->cpu_exec_interrupt()

TCG生成与处理

```
int cpu_exec(CPUState *cpu)
{
    CPUClass *cc = CPU_GET_CLASS(cpu);

    current_cpu = cpu;

    cc->cpu_exec_enter(cpu);

    init_delay_params(&sc, cpu);

    while (!cpu_handle_exception(cpu, &ret)) {
        TranslationBlock *last_tb = NULL;
        int tb_exit = 0;

        while (!cpu_handle_interrupt(cpu, &last_tb)) {
            uint32_t cflags = cpu->cflags_next_tb;
            TranslationBlock *tb;
            if (cflags == -1) {
                cflags = curr_cflags();
            } else {
                cpu->cflags_next_tb = -1;
            }

            tb = tb_find(cpu, last_tb, tb_exit, cflags);
            cpu_loop_exec_tb(cpu, tb, &last_tb, &tb_exit);
            /* Try to align the host and virtual clocks
               if the guest is in advance */
            align_clocks(&sc, cpu);
        }
    }

    cc->cpu_exec_exit(cpu);
    return ret;
}
```

TCG 原理

TCG 是什么?

Tiny Code Generator (TCG)

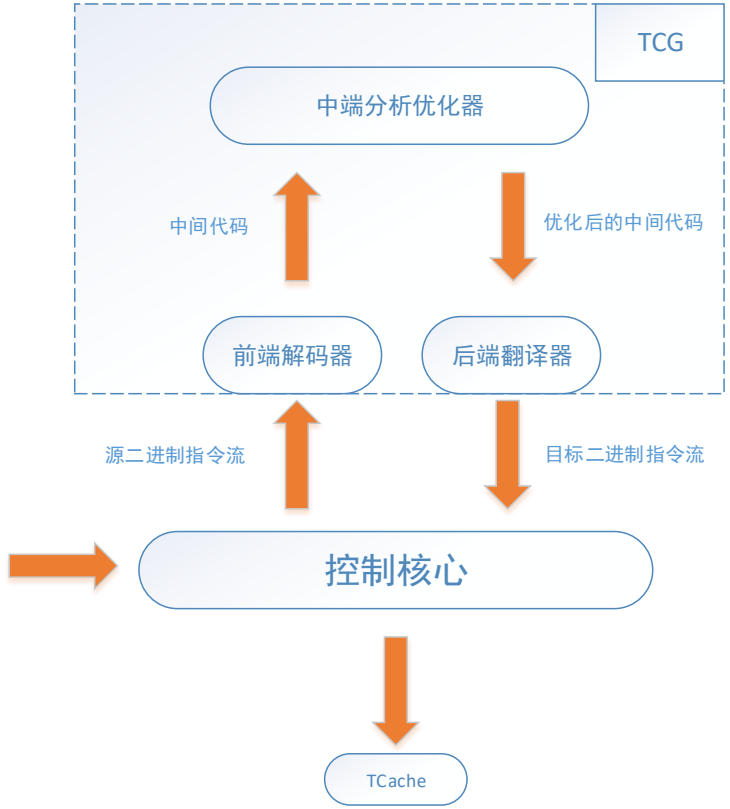
将源处理器机器代码(如RISC-V机器代码)转换为虚拟机运行所需的机器代码块(如x86机器代码块)

二进制翻译：

源指令流：RISC-V机器码

中间代码：TCG IR

目标指令流：X86机器码



TCG 转换示例

用户态运行 risc-v hello world为例

文件hello.s

```
.global _start      # Provide program starting address to linker

# Setup the parameters to print hello world
# and then call Linux to do it.

_start: addi  a0, x0, 1      # 1 = StdOut
        la    a1, helloworld # load address of helloworld
        addi  a2, x0, 13     # length of our string
        addi  a7, x0, 64     # linux write system call
        ecall                # Call linux to output the string

# Setup the parameters to exit the program
# and then call Linux to do it.

        addi    a0, x0, 0   # Use 0 return code
        addi    a7, x0, 93  # Service command code 93 terminates
        ecall               # Call linux to terminate the program

.data
helloworld:      .ascii "Hello World!\n"
```

编译方法:

```
$riscv-nuclei-elf-as -o hello.o hello.s
$riscv-nuclei-elf-ld -o hello hello.o
```

运行结果:

```
$qemu-riscv32 hello
Hello World!
```

```
qemu-riscv32 -d help
Log items (comma separated):
out_asm      show generated host assembly code for each compiled TB
in_asm       show target assembly code for each compiled TB
op           show micro ops for each compiled TB
op_opt       show micro ops after optimization
op_ind       show micro ops before indirect lowering
int          show interrupts/exceptions in short format
```

参考:RISC-V Assembly Programmer's Manual

TCG 转换示例

$qemu-riscv32 -d in_asm hello

```
----------------
IN:
0x00010074:  00100513      addi      a0,zero,1
0x00010078:  00001597      auipc     a1,4096          # 0x11078
0x0001007c:  02058593      addi      a1,a1,32
0x00010080:  00d00613      addi      a2,zero,13
0x00010084:  04000893      addi      a7,zero,64
0x00010088:  00000073      ecall

Hello World!
----------------
IN:
0x0001008c:  00000513      mv        a0,zero
0x00010090:  05d00893      addi      a7,zero,93
0x00010094:  00000073      ecall
```

$qemu-riscv32 -d op hello

```
---- 0001007c
 mov_i32 tmp0,x11/a1
 add_i32 tmp0,tmp0,$0x20
 mov_i32 x11/a1,tmp0

 ---- 00010080
 mov_i32 tmp0,$0x0
 add_i32 tmp0,tmp0,$0xd
 mov_i32 x12/a2,tmp0

 ---- 00010084
 mov_i32 tmp0,$0x0
 add_i32 tmp0,tmp0,$0x40
 mov_i32 x17/a7,tmp0
```

RISC-V to TCG IR

TCG 转换示例

TCG IR to Host Code

$qemu-riscv32 -d out_asm hello

源二进制流

```
0x0001007c:  02058593      addi      a1,a1,32
0x00010080:  00d00613      addi      a2,zero,13
0x00010084:  04000893      addi      a7,zero,64
0x00010088:  00000073      ecall
```

未优化的TCG IR

```
 ---- 00010080
mov_i32 tmp0,$0x0
add_i32 tmp0,tmp0,$0xd
mov_i32 x12/a2,tmp0
```

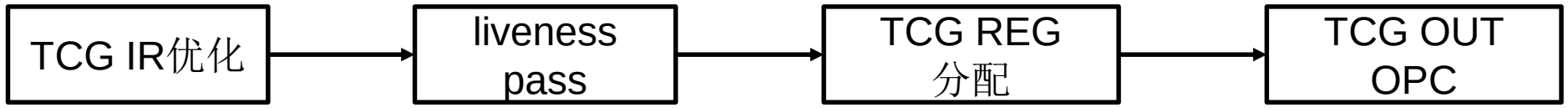
```
OUT: [size=88]
  -- guest addr 0x00010074 + tb prologue
0x7f56380000c0:  8b 5d f0                movl     -0x10(%rbp), %ebx
0x7f56380000c3:  85 db                   testl    %ebx, %ebx
0x7f56380000c5:  0f 8c 34 00 00 00       jl       0x7f56380000ff
0x7f56380000cb:  c7 45 28 01 00 00 00    movl     $1, 0x28(%rbp)
  -- guest addr 0x0001007c
0x7f56380000d2:  c7 45 2c 98 10 01 00    movl     $0x11098, 0x2c(%rbp)
  -- guest addr 0x00010080
0x7f56380000d9:  c7 45 30 0d 00 00 00    movl     $0xd, 0x30(%rbp)
  -- guest addr 0x00010084
0x7f56380000e0:  c7 45 44 40 00 00 00    movl     $0x40, 0x44(%rbp)
  -- guest addr 0x00010088
0x7f56380000e7:  c7 85 94 05 00 00 88 00 movl     $0x10088, 0x594(%rbp)
0x7f56380000ef:  01 00
0x7f56380000f1:  48 8b fd                movq     %rbp, %rdi
0x7f56380000f4:  be 08 00 00 00          movl     $8, %esi
0x7f56380000f9:  ff 15 11 00 00 00       callq    *0x11(%rip)
0x7f56380000ff:  48 8d 05 3d ff ff ff    leaq     -0xc3(%rip), %rax
0x7f5638000106:  e9 0d ff ff ff          jmp      0x7f5638000018
```

优化的TCG IR

```
---- 0001007c
 mov_i32 x11/a1,$0x11098  sync: 0  dead: 0 1
pref=0xffff

 ---- 00010080
 mov_i32 x12/a2,$0xd  sync: 0  dead: 0 1  pref=0xffff

 ---- 00010084
 mov_i32 x17/a7,$0x40  sync: 0  dead: 0 1  pref=0xffff
```

TCG 转换示例

Guest Code to TCG IR to Host Code



RISC-V

```
0x00010080:  00d00613      addi      a2,zero,13
```

TCG IR

```
 ---- 00010080
 mov_i32 tmp0,$0x0
 add_i32 tmp0,tmp0,$0xd
 mov_i32 x12/a2,tmp0
……
```

优化后

-op_opt输出

```
---- 00010080
 mov_i32 x12/a2,$0xd  sync: 0  dead: 0 1  pref=0xffff
```

X86_64

```
-- guest addr 0x00010080
0x7f56380000d9:  c7 45 30 0d 00 00 00      movl      $0xd, 0x30(%rbp)
```

TCG IR

定义于:include/tcg/tcg-opc.h
7种IR类别

IR类别	示例(32/64)
predefined ops	discard,set_label, call, br,etc.
load/store	ld,st ,etc.
arith	add, sub, mul, div,etc.
shifts/rotates	shl, shr, sar,bswap ,etc.
size changing ops	ext_i32_i64, extrl_i64_i32,extrh_i64_i32 ,etc.
QEMU specific	insn_start, exit_tb, goto_tb, qemu_ld, qemu_st ,etc.
Host vector support	mov_vec, ld_vec, st_vec,add_vec,not_vec ,etc.

IR示例：

```
---- 00010080
mov_i32 tmp0,$0x0
add_i32 tmp0,tmp0,$0xd
mov_i32 x12/a2,tmp0
```

定义格式：

```
typedef enum TCGOpcode {
#define DEF(name, oargs, iargs, cargs, flags) INDEX_op_ ##
name,
#include "tcg/tcg-opc.h"
#undef DEF
    NB_OPS,
} TCGOpcode;
```

name: 指令名称
oargs: output 参数个数
iargs: input 参数个数
cargs: const 参数个数
flags: flag

源码IR示例：

```
/* arith */
DEF(add_i32, 1, 2, 0, 0)
DEF(sub_i32, 1, 2, 0, 0)
DEF(mul_i32, 1, 2, 0, 0)
DEF(div_i32, 1, 2, 0, IMPL(TCG_TARGET_HAS_div_i32))
```

INDEX_op_add_i32
INDEX_op_sub_i32
INDEX_op_mul_i32
INDEX_op_div_i32

TCG转换

主要调用栈:

```
cpu_loop(CPURISCVState *env)
cpu_exec(CPUState *cpu)
TranslationBlock *tb_find(……)
    TranslationBlock *tb_gen_code(……)
        void gen_intermediate_code(……)
            translator_loop(const TranslatorOps *ops, ……)
        tcg_gen_code(TCGContext *s, TranslationBlock *tb)
```

(用户态)

arch独立

TCG中重要类型:

```
struct TranslationBlock
struct TranslatorOps
struct TCGContext → struct TCGOp
```

TCG转换之类型

文件:include/exec/cpu-all.h

```
struct TranslationBlock {
    target_ulong pc;
    target_ulong cs_base; /* CS base for this block */
    uint32_t flags; enerated */
    uint32_t cflags;    /* compile flags */
    uint32_t trace_vcpu_dstate;
    uint16_t size;
    uint16_t icount;
    struct tb_tc tc;
    uintptr_t page_next[2];
    tb_page_addr_t page_addr[2];
    QemuSpin jmp_lock;
    uint16_t jmp_reset_offset[2]; /* offset of original jump target */
    uintptr_t jmp_target_arg[2];  /* target address or offset */
    uintptr_t jmp_list_head;
    uintptr_t jmp_list_next[2];
    uintptr_t jmp_dest[2];
};
```

TranslationBlock主要结构
pc
cs_base
icount
orig_tb
jmp_lock
jmp_list_head
jmp_list_next[2]
jmp_dest[2]

TCG转换之类型

文件:include/tcg/tcg.h

```
struct TCGContext {
    ……
    GHashTable *const_table[TCG_TYPE_COUNT];
    TCGTempSet free_temps[TCG_TYPE_COUNT * 2];
    TCGTemp temps[TCG_MAX_TEMPS];
    /* globals first, temps after */
    ……
    QTAILQ_HEAD(, TCGOp) ops, free_ops;
    ……
    TCGTemp *reg_to_temp[TCG_TARGET_NB_REGS];
};
```

struct TCGTemp

```
typedef struct TCGOp {
    TCGOpcode opc   : 8;        /*  8 */

    /* Parameters for this opcode.  See below.  */
    unsigned param1 : 4;        /* 12 */
    unsigned param2 : 4;        /* 16 */

    /* Lifetime data of the operands.  */
    unsigned life   : 16;       /* 32 */

    /* Next and previous opcodes.  */
    QTAILQ_ENTRY(TCGOp) link;
#ifdef CONFIG_PLUGIN
    QSIMPLEQ_ENTRY(TCGOp) plugin_link;
#endif

    /* Arguments for the opcode.  */
    TCGArg args[MAX_OPC_PARAM];

    /* Register preferences for the output(s).  */
    TCGRegSet output_pref[2];
} TCGOp;
```

TCG转换之类型

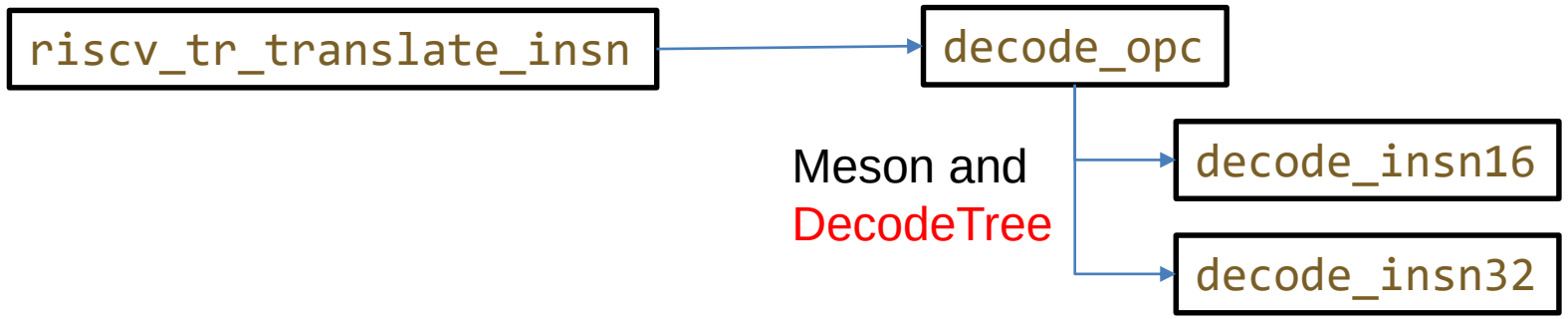
```
void gen_intermediate_code(……)
translator_loop(const TranslatorOps *ops, ……)
```

target/arch/translate.c

```
struct TranslatorOps
    void (*init_disas_context)
    void (*tb_start)
    void (*insn_start)
    bool (*breakpoint_check)
    void (*translate_insn)
    void (*tb_stop)
    void (*disas_log)
```

文件：target/riscv/translate.c

```
static const TranslatorOps riscv_tr_ops = {
    .init_disas_context = riscv_tr_init_disas_context,
    .tb_start           = riscv_tr_tb_start,
    .insn_start         = riscv_tr_insn_start,
    .breakpoint_check   = riscv_tr_breakpoint_check,
    .translate_insn     = riscv_tr_translate_insn,
    .tb_stop            = riscv_tr_tb_stop,
    .disas_log          = riscv_tr_disas_log,
};
```



文件：target/riscv/meson.build

```
gen32 = [
  decodetree.process('insn16.decode', extra_args: [dir / 'insn16-32.decode', '--static-decode=decode_insn16', '--insnwidth=16']),
  decodetree.process('insn32.decode', extra_args: '--static-decode=decode_insn32'),
]
```

TCG转换之过程

TCG环境初始化:

```
static struct TCGCPUOps riscv_tcg_ops = {
    .initialize = riscv_translate_init,
    ……
}
```

转换过程:

```
translator_loop(const TranslatorOps *ops, ……)
```

文件：target/arch/translate.c

```
static const TranslatorOps riscv_tr_ops = {
    .init_disas_context = riscv_tr_init_disas_context,
    .tb_start           = riscv_tr_tb_start,
    .insn_start         = riscv_tr_insn_start,
    .breakpoint_check   = riscv_tr_breakpoint_check,
    .translate_insn     = riscv_tr_translate_insn,
    .tb_stop            = riscv_tr_tb_stop,
    .disas_log          = riscv_tr_disas_log,
};
```

全局变量的内存映射转换:

```
/* global register indices */
static TCGv cpu_gpr[32], cpu_pc, cpu_vl;
static TCGv_i64 cpu_fpr[32]; /* assume F and D extensions */
static TCGv load_res;
static TCGv load_val;
```

```
typedef struct DisasContext {
    ……
    target_ulong pc_succ_insn;
    uint32_t opcode;
    uint32_t misa;
    ……
} DisasContext;
```

translator_loop

- init_disas_context 初始DisasContext
- gen_tb_start(db->tb) and tb_start(db, cpu) 开始翻译
- insn_start(db, cpu) → INDEX_op_insn_start
- breakpoint_check → exception debug
- translate_insn → decode_opc
- tb_stop(db, cpu) and gen_tb_end
- disas_log

TCG转换之过程(无优化)

decode_insn32匹配

文件：target\riscv\insn_trans\trans_rvi.inc.c

源二进制流

rd rs1 imm

```
0x00010080:  00d00613      addi      a2,zero,13
0x00010084:  04000893      addi      a7,zero,64
```

未优化的TCG IR

```
---- 00010080
mov_i32 tmp0,$0x0
add_i32 tmp0,tmp0,$0xd
mov_i32 x12/a2,tmp0
```

```
static bool trans_addi(DisasContext *ctx, arg_addi *a)
{
    return gen_arith_imm_fn(ctx, a, &tcg_gen_addi_tl);
}
```

target\riscv\translate.c

```
static bool gen_arith_imm_fn(DisasContext *ctx, arg_i *a,
                             void (*func)(TCGv, TCGv, target_long))
{
    TCGv source1;
    source1 = tcg_temp_new();

    gen_get_gpr(source1, a->rs1);

    (*func)(source1, source1, a->imm);

    gen_set_gpr(a->rd, source1);
    tcg_temp_free(source1);
    return true;
}
```

tmp0

mov_i32 tmp0,$0x0

add_i32 tmp0,tmp0,$0xd

mov_i32 x12/a2,tmp0

```
#define tcg_gen_addi_tl tcg_gen_addi_i32
#define tcg_gen_movi_tl tcg_gen_movi_i32
```

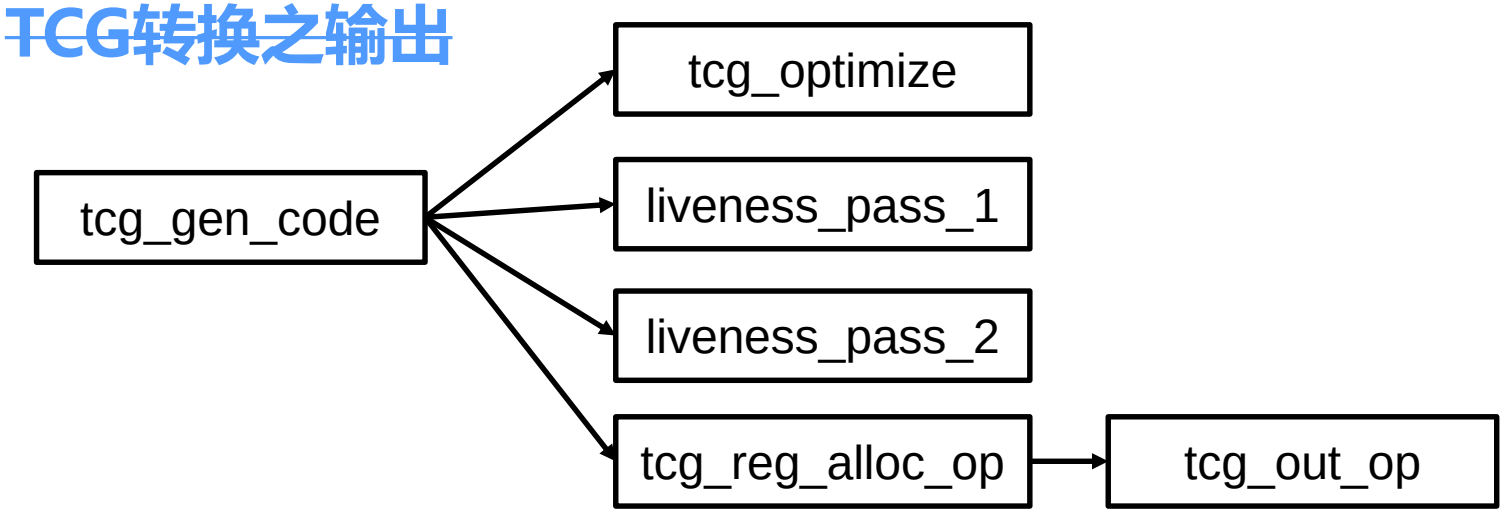
TCG 前端/后端

```
static inline void gen_get_gpr(TCGv t, int reg_num)
{
    if (reg_num == 0) {
        tcg_gen_movi_tl(t, 0);
    } else {
        tcg_gen_mov_tl(t, cpu_gpr[reg_num]);
    }
}
void tcg_gen_addi_i32(TCGv_i32 ret, TCGv_i32 arg1, int32_t arg2)
{
    /* some cases can be optimized here */
    if (arg2 == 0) {
        tcg_gen_mov_i32(ret, arg1);
    } else {
        tcg_gen_add_i32(ret, arg1, tcg_constant_i32(arg2));
    }
}
```

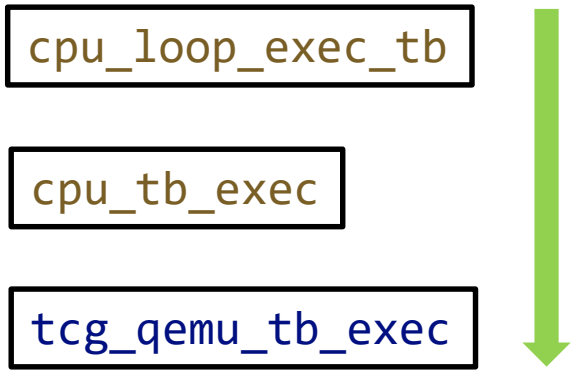
INDEX_op_mov_i32

```
static inline void tcg_gen_mov_i32(TCGv_i32 ret, TCGv_i32 arg)
{
    if (ret != arg) {
        tcg_gen_op2_i32(INDEX_op_mov_i32, ret, arg);
    }
}
static inline void tcg_gen_add_i32(TCGv_i32 ret, TCGv_i32 arg1, TCGv_i32 arg2)
{
    tcg_gen_op3_i32(INDEX_op_add_i32, ret, arg1, arg2);
}
```

TCG转换之输出



TCG执行




```
int tcg_gen_code(TCGContext *s, TranslationBlock *tb)
{
……
    switch (opc) {
        case INDEX_op_mov_i32:
        case INDEX_op_mov_i64:
        case INDEX_op_mov_vec:
            tcg_reg_alloc_mov(s, op);
            break;
        case INDEX_op_dup_vec:
            tcg_reg_alloc_dup(s, op);
            break;
        case INDEX_op_insn_start:
……
}
```

i386 Opcode示例：

```
#define OPC_MOVB_EvGv    (0x88)       /* stores, more or less */
#define OPC_MOVL_EvGv    (0x89)       /* stores, more or less */
#define OPC_MOVL_GvEv    (0x8b)       /* loads, more or less */
#define OPC_MOVB_EvIz    (0xc6)
#define OPC_MOVL_EvIz    (0xc7)
#define OPC_MOVL_Iv      (0xb8)
```

位于:tcg/i386/tcg-target.c.inc文件中

下节课内容：

CPU虚拟化

- ~~RISCV CPU实现分析~~
- ~~新RISCV CPU建立~~
- ~~TCG原理与指令翻译~~
- DecodeTree与NICE指令添加
- CSR寄存器实现添加

谢谢

wangjunqiang@iscas.ac.cn