Virtual Machine Homework 1

組員：

- B04703001 資工三 蔡明宏
- B04705001 資工四 陳約廷

Download & Execute

檔案：`vm_hw_b04703001_b04705001.tar`

- `Assignment1`：第一大題的整包 QEMU
- `Assignment2`：第二大題的整包 QEMU
- `report.pdf`：Design & Implementation

‼️ 進去 `Assignment*/` 可以執行：

```
./configure --target-list=aarch64-linux-user
make

會有執行檔：
aarch64-linux-user/qemu-aarch64
```

‼️ Assignment 1 的輸出是在 stderr 裡面（見 `linux-user/syscall.c` 裡面的 `my_enc`

Assignment 1

首先瞭解 QEMU 執行結構

由 Guest Code 經由 `gen_intermediate_code()` 之後轉換成 TCG (Tiny Code Generator) IR (Intermediate Representation)，TCG IR 再經由 `tcg_gen_code()` 轉換成 Host Machine 上的 Native Code.

- Front-End: $Source\ ISA \rightarrow TCG\ IR$
- Back-End: $TCG\ IR \rightarrow Target\ ISA$

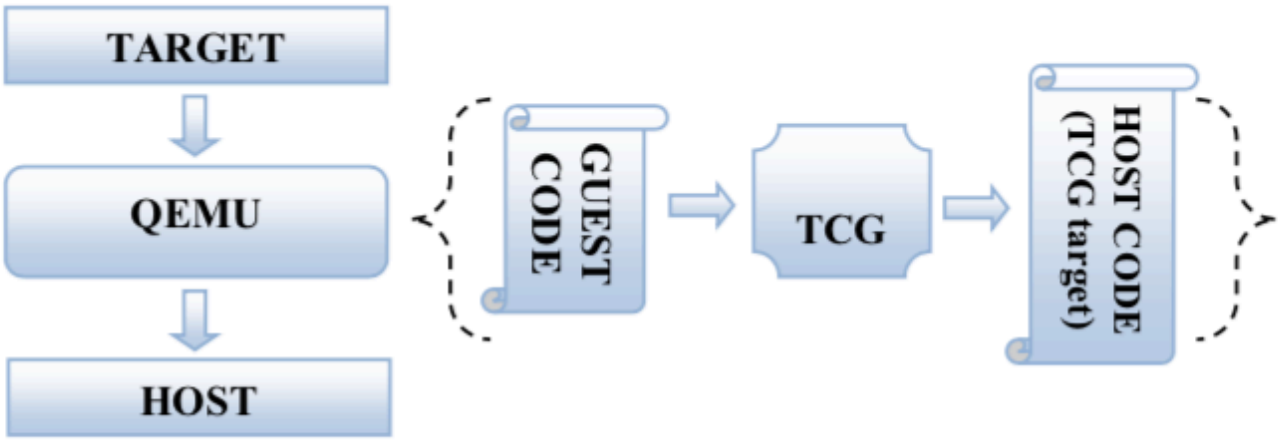

Figure 7.1: Use of term 'Target'

QEMU 對前端的部分有 `helper_function` 這個 API 對當前 CPU 環境 `cpu-env` 以及當下的變數進行操作。

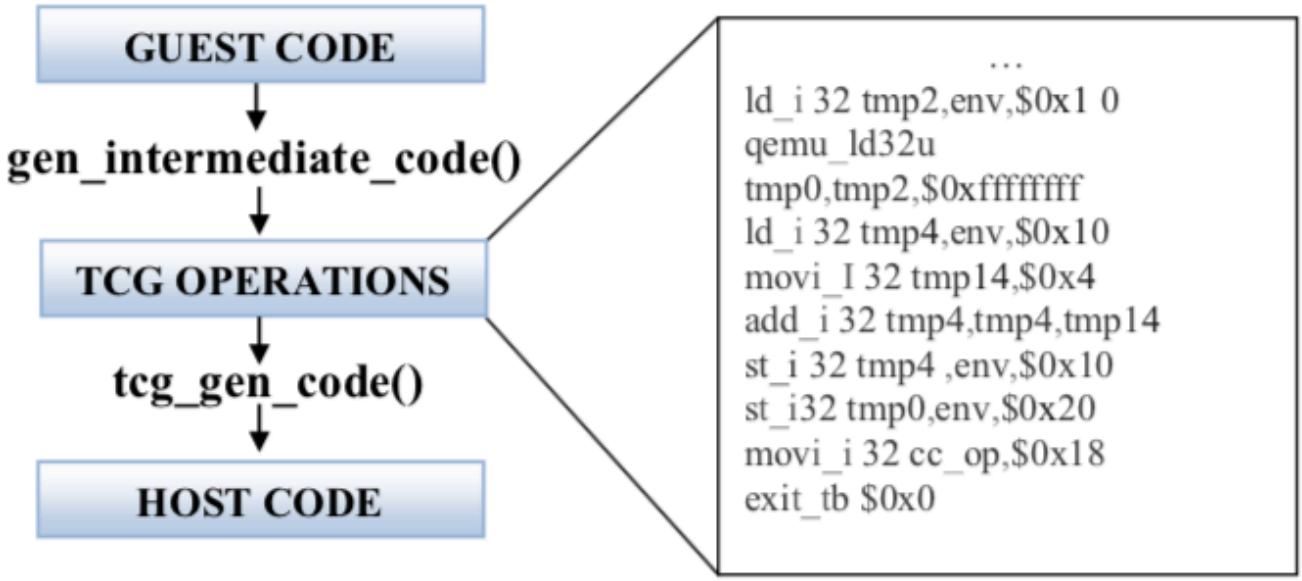

Figure 7.6: Dynamic translation – showing TCG ops

Trace to branch

因為要模擬的 source ISA 為 ARMv8 ，因此只需要修改 `/target/arm/` 底下中的部分，不需要修改 `target/` 中其他 ISA 。

在 `translate-a64.c` 之下……

- `gen_intermediate_code_a64` -> `disas_a64_insn` -> `disas_b_exc_sys` ：進行 branch 判斷
    - `disas_uncond_b_imm` ：Unconditional branch (immediate)
    - `disas_comp_b_imm` ：Compare & branch (immediate)
    - `disas_test_b_imm` ：Test & branch (immediate)
    - `disas_cond_b_imm` ：Conditional branch (immediate)
    - `disas_uncond_b_reg` ：Unconditional branch (register)

這些 branch 方式只分為 branch with register 與其他種 branch 兩種。見以下 helper function 對這個的 parsing 。

全域變數 - HashMap

`include/qemu-common.h` 之中：

```
typedef struct HashMap HashMap;
struct HashMap{
    Node nodeCnt[3][100000];
    int size[3];
    void (*initialize)(struct HashMap *);
    void (*add)(struct HashMap *, uint64_t, int);
    void (*printMap)(struct HashMap* self);
    int (*compare)(const void * a, const void *b);
};

extern HashMap hashMap;
```

在 `linux-user/main.c` 之中 initialize 這個資料結構，並且裡頭的 add 函式可以被 helper function 呼叫。

`printMap` 之中用 `getenv()` 得到的參數輸出結果

Helper Function - Register

`disas_uncond_b_reg` 是以 register 做 indirect branch ，所以這裡與 helper function 要互動的是全域變數 `cpu_env` 以及擷取 instruction 中 `rn` 來去 access `lr` register 做 link。

`void HELPER(log_reg_branch)(CPUARMState* env, uint64_t pc, uint64_t rn);`

Helper Function - No-Register

其他的 branching ，會透過 `gen_goto_tb` 來產生朝下一個 TranslationBlock 的做 link 。這裡可以創造 `TCGv_i64` 這樣的新記憶體來抓 sourcePC 與 targetPC ，並送進 helperfunction 之中。最後要記得 freememory 。

藉由 helper_function ，在 `gen_goto_tb` 擷取要進行 branch 的 `source_PC` 以及 `target_PC` 。

這裡在 `gen_goto_tb` 擴增一個參數來記錄從那些 branching condition 呼叫過來的 branch 是 taken 還是 not_taken ，把它記錄在全域變數裡面。

要注意的是 `gen_intermediate_code` 之中，如果一個 BasicBlock 的指令數量到該 page 可以裝下的指令上限， QEMU 也會呼叫 `gen_goto_tb` ，要把這個操作傳過來的設為無效的 branch 。

`void HELPER(log_imm_branch){uint_64_t pc, uint64_t addr, uint64_t is_taken);`

Assignment 2

使用 `aarch64-linux-gnu-objdump` 可以把他的 ARM v8 code 轉譯出來：

- `aarch64-linux-gnu-objdump -d encr-vm > encr-vm.dump`

打開來可以看到 `encry` 的函數長得很像 encrypt 該有的東西。

```
$ cat release/encr-vm.dump | grep encry
0000000000400658 <encry>:
  400678:   14000021    b   4006fc <encry+0xa4>
  400688:   540001c0    b.eq    4006c0 <encry+0x68>  // b.none
  4006bc:   1400000d    b   4006f0 <encry+0x98>
  400708:   54fffbab    b.lt    40067c <encry+0x24>  // b.tstop
  40078c:   97ffffb3    bl  400658 <encry>
```

- 在呼叫 `encry` 之處為 `ret` ，為 unconditional branch
- 承 Assignment 1 ，在 `disas_uncond_b_imm` 中多增加一個條件來跳過這一行
- s->pc 為不跳躍時的下一個 instruction ，因此直接判斷

```
if ( s->pc-4 == 0x40078c )
    addr = s->pc;
```

❗ 為了不影響 Assignment 1 執行的結果，因此分成兩個 QEMU 檔案來上傳。

Reference

- Explains QEMU structure & execution flow: "Qemu Detailed Study"
- Helper function API: "QEMU: Call a Custom Function from TCG"