Cpu-exec.c:

```
int cpu_exec(CPUState *env) {
        Создается TranslationBlock;
        Выполнение сгенерированного кода;
        Если код не сгенерирован, то вызывается
        cpu_exec_nocache(env, insns_left, tb);
}
```

Cpu-exec.c:

```
void cpu_exec_nocache(CPUState *env, int max_cycles, TranslationBlock *orig_tb) {
        /* generate code */
        tb = tb_gen_code(env, orig_tb->pc, orig_tb->cs_base, orig_tb->flags, max_cycles);
        env->current_tb = tb;
         /* execute the generated code */
        next_tb = tcg_qemu_tb_exec(env, tb->tc_ptr);

        /* Restore PC.  */
        cpu_pc_from_tb(env, tb);
    }
```

Exec.c:

```
TranslationBlock *tb_gen_code(CPUState *env,  target_ulong pc, target_ulong cs_base,
                              int flags, int cflags) {
        /* установка указателя на транслируемый код, установка флагов */
        tb->tc_ptr = code_gen_ptr;
        ……..
        cpu_gen_code(env, tb, &code_gen_size);
        /* add a new TB and link it to the physical page tables. */
        tb_link_page(tb, phys_pc, phys_page2);
}
```

Translate-all.c:

```
int cpu_gen_code(CPUState *env, TranslationBlock *tb, int *gen_code_size_ptr) {
        tcg_func_start(s);
        gen_intermediate_code(env, tb);
         /* generate machine code */
        gen_code_size = tcg_gen_code(s, gen_code_buf);
        /* logging for OUT code */
        log_disas(tb->tc_ptr, *gen_code_size_ptr);
}
```

search_pc = 0

Target-arm/translate.c:

```
void gen_intermediate_code_internal(CPUState *env, TranslationBlock *tb, search_pc) {
        Do {    ...
                disas_arm_insn(env, dc);
        } while (Translation stops when a conditional branch is encountered. Otherwise the
subsequent code could get translated several times.);
        /* logging IN code */
        log_target_disas(pc_start, dc->pc - pc_start, dc->thumb);
}
```

Target-arm/translate.c:

```
disas_arm_insn(CPUState * env, DisasContext *s) {
        int cond, insn, val, op1, i, shift, rm, rs, rn, rd, sh; //parameters
        TCGv tmp, tmp2, tmp3, addr; //temporary regs
        insn = ldl_code(s->pc); //binary instruction
        cond = insn >> 28; //condition
        /* instruction encoding */
        if ((insn & 0x0f900000) == 0x01000000 && (insn & 0x00000090) != 0x00000090) {
                op1 = (insn >> 21) & 3; //operand
                sh = (insn >> 4) & 0xf; //shift
                rm = insn & 0xf; //dest register = immed value
                switch (sh) {
                case 0x0: /* move program status register encoding*/
                 if (op1 & 1) { /* PSR = reg    msr */
                        tmp = load_reg(s, rm); //Create a new tmp, set it to the value of a CPU register.
                        //set PSR function
                        gen_set_psr(s, msr_mask(env, s, (insn >> 16) & 0xf, i), i, tmp);
                 } else { /* reg = PSR  mrs */
                        rd = (insn >> 12) & 0xf; //dest register
                        if (op1 & 2) {
                                if (IS_USER(s)) goto illegal_op;
                                tmp = load_cpu_field(spsr); //set name of cpu field = spsr
                        } else {
                                tmp = tcg_temp_new_i32();
                                gen_helper_cpsr_read(tmp); // generate call-function
                        }
                        store_reg(s, rd, tmp); //store
                 }
                 break;
        ……..........
        ……..........
}
```