in arch/arm/kernel/entry-armv.S

```
/*
 * Undef instr entry dispatcher
 * Enter in UND mode, spsr = SVC/USR CPSR, lr = SVC/USR PC
 */
	vector_stub	und, UND_MODE

	.long	__und_usr			@  0 (USR_26 / USR_32)
	.long	__und_invalid		@  1 (FIQ_26 / FIQ_32)
	.long	__und_invalid		@  2 (IRQ_26 / IRQ_32)
	.long	__und_svc			@  3 (SVC_26 / SVC_32)
	.long	__und_invalid		@  4
	.long	__und_invalid		@  5
	.long	__und_invalid		@  6
	.long	__und_invalid		@  7
	.long	__und_invalid		@  8
	.long	__und_invalid		@  9
	.long	__und_invalid		@  a
	.long	__und_invalid		@  b
	.long	__und_invalid		@  c
	.long	__und_invalid		@  d
	.long	__und_invalid		@  e
	.long	__und_invalid		@  f
```

__und_usr is the handler for user mode application trigger undefined-instrunction,

and __und_svc is supervisor mode kernel code trigger undefined-instrunction.

无论运行arm(32-bit) or thumb(16-bit)的user mode或supervisor mode，都会归结到**__und_fault**

__und_fault:

```
    @ Correct the PC such that it is pointing at the instruction
    @ which caused the fault.  If the faulting instruction was ARM
    @ the PC will be pointing at the next instruction, and have to
    @ subtract 4.  Otherwise, it is Thumb, and the PC will be
    @ pointing at the second half of the Thumb instruction.  We
    @ have to subtract 2.
    ldr    r2, [r0, #S_PC]
    sub  r2, r2, r1
    str    r2, [r0, #S_PC]
    b      do_undefinstr
ENDPROC(__und_fault)
```

in arch/arm/kernel/traps.c

```
asmlinkage void __exception do_undefinstr(struct pt_regs *regs)
{
```

```
        unsigned int instr;
        siginfo_t info;
        void __user *pc;

        pc = (void __user *)instruction_pointer(regs);                    (0)

        if (processor_mode(regs) == SVC_MODE) {
#ifdef CONFIG_THUMB2_KERNEL                                                (1)
                if (thumb_mode(regs)) {
                        instr = __mem_to_opcode_thumb16(((u16 *)pc)[0]);
                        if (is_wide_instruction(instr)) {
                                u16 inst2;
                                inst2 = __mem_to_opcode_thumb16(((u16 *)pc)[1]);
                                instr = __opcode_thumb32_compose(instr, inst2);
                        }
                } else
#endif
                        instr = __mem_to_opcode_arm(*(u32 *) pc);
        } else if (thumb_mode(regs)) {                                     (2)
                if (get_user(instr, (u16 __user *)pc))
                        goto die_sig;
                instr = __mem_to_opcode_thumb16(instr);
                if (is_wide_instruction(instr)) {
                        unsigned int instr2;
                        if (get_user(instr2, (u16 __user *)pc+1))
```

```
                goto die_sig;
            instr2 = __mem_to_opcode_thumb16(instr2);
            instr = __opcode_thumb32_compose(instr, instr2);
        }
    } else {                                                    (3)
        if (get_user(instr, (u32 __user *)pc))
            goto die_sig;
        instr = __mem_to_opcode_arm(instr);                     (4)
    }

    if (call_undef_hook(regs, instr) == 0)                      (5)
        return;

die_sig:
#ifdef CONFIG_DEBUG_USER
    if (user_debug & UDBG_UNDEFINED) {
        printk(KERN_INFO "%s (%d): undefined instruction: pc=%p\n",
            current->comm, task_pid_nr(current), pc);
        __show_regs(regs);
        dump_instr(KERN_INFO, regs);
    }
#endif

    info.si_signo = SIGILL;                                     (6)
    info.si_errno = 0;
```

```
        info.si_code  = ILL_ILLOPC;
        info.si_addr  = pc;

        arm_notify_die("Oops - undefined instruction", regs, &info, 0, 6);
}
```

(0)

pc指向产生undefined instruction的address。

(1)

in Gr2 / Gs2 LSP,

CONFIG_THUMB2_KERNEL is not set

(2)

Gr2 / Gs2 LSP并不运行与thumb mode。

(3)

由pc取出undefined instruction.

(4)

在LSP config中，__mem_to_opcode_arm等于没有任何作用。

(5)

Linux kernel给了其他module一个机会，让它们去辨认该条“undefined instruction”.如果其被“认领”，也就是被其他module处理了，则返回0，kernel就象什么事没发生一样。这是一个很强大的机制，比如可以用来emulate a new instruction(the instruction has not implemented in the old arch)

(6)

设置SIGILL signal,从该exception返回后，产生该exception的process将被该signal杀死。

```
1.  static LIST_HEAD(undef_hook);
2.  static DEFINE_RAW_SPINLOCK(undef_lock);
3.
4.  void register_undef_hook(struct undef_hook *hook)
5.  {
6.          unsigned long flags;
7.
8.          raw_spin_lock_irqsave(&undef_lock, flags);
9.          list_add(&hook->node, &undef_hook);
10.         raw_spin_unlock_irqrestore(&undef_lock, flags);
11. }
12.
13. void unregister_undef_hook(struct undef_hook *hook)
14. {
15.         unsigned long flags;
16.
17.         raw_spin_lock_irqsave(&undef_lock, flags);
18.         list_del(&hook->node);
19.         raw_spin_unlock_irqrestore(&undef_lock, flags);
20. }
21.
22. static int call_undef_hook(struct pt_regs *regs, unsigned int instr)
23. {
24.         struct undef_hook *hook;
25.         unsigned long flags;
26.         int (*fn)(struct pt_regs *regs, unsigned int instr) = NULL;
27.
28.         raw_spin_lock_irqsave(&undef_lock, flags);
29.         list_for_each_entry(hook, &undef_hook, node)
30.                 if ((instr & hook->instr_mask) == hook->instr_val &&
31.                     (regs->ARM_cpsr & hook->cpsr_mask) == hook->cpsr_val)
32.                         fn = hook->fn;
33.         raw_spin_unlock_irqrestore(&undef_lock, flags);
34.
35.         return fn ? fn(regs, instr) : 1;
36. }
```

in arch/arm/include/asm/traps.h

struct undef_hook {

```
        struct list_head node;
        u32 instr_mask;
        u32 instr_val;
        u32 cpsr_mask;
        u32 cpsr_val;
        int (*fn)(struct pt_regs *regs, unsigned int instr);
};
```

由于SWP/SWPB instructions已经deprecates，所以armv7 CPU硬件上没有实现，但如果要支持该指令，则必须用软件emulate；否则运行含有该instructions的application将会因为收到"undefined instruction" exception而异常终止。

CONFIG_SWP_EMULATE=y in Gr2 / Gs2 LSP

in arch/arm/kernel/swp_emulate.c

in arch/arm/mm/Kconfig

```
config SWP_EMULATE
        bool "Emulate SWP/SWPB instructions" if !SMP
        depends on CPU_V7
        default y if SMP
        select HAVE_PROC_CPU if PROC_FS
        help
          ARMv6 architecture deprecates use of the SWP/SWPB instructions.
```

ARMv7 multiprocessing extensions introduce the ability to disable
these instructions, triggering an undefined instruction exception
when executed. Say Y here to enable software emulation of these
instructions for userspace (not kernel) using LDREX/STREX.
Also creates /proc/cpu/swp_emulation for statistics.

In some older versions of glibc [<=2.8] SWP is used during futex
trylock() operations with the assumption that the code will not
be preempted. This invalid assumption may be more likely to fail
with SWP emulation enabled, leading to deadlock of the user
application.

NOTE: when accessing uncached shared regions, LDREX/STREX rely
on an external transaction monitoring block called a global
monitor to maintain update atomicity. If your system does not
implement a global monitor, this option can cause programs that
perform SWP operations to uncached memory to deadlock.

If unsure, say Y.

root@granite2:~# cat /proc/cpu/swp_emulation

Emulated SWP:0

Emulated SWPB:0

Aborted SWP{B}:0