start_kernel()

|

|

\|/

setup_arch() in arch/arm/kernel/setup.c

|

|

\|/

paging_init() in arch/arm/mm/mmu.c

|

|

\|/

devicemaps_init() in arch/arm/mm/mmu.c

|

|

\|/

early_trap_init()

```
1.  static void __init devicemaps_init(const struct machine_desc *mdesc)
2.  {
3.      ......
4.
5.          /*
6.           * Allocate the vector page early.
7.           */
8.          vectors = early_alloc(PAGE_SIZE * 2);         ①
9.
10.         early_trap_init(vectors);                     ②
11.
12.     ......
13.
14.         /*
15.          * Create a mapping for the machine vectors at the high-vectors
16.          * location (0xffff0000).  If we aren't using high-vectors, also
17.          * create a mapping at the low-vectors virtual address.
18.          */
19.         map.pfn = __phys_to_pfn(virt_to_phys(vectors));
20.         map.virtual = 0xffff0000;                ③
21.         map.length = PAGE_SIZE;
22. #ifdef CONFIG_KUSER_HELPERS
23.         map.type = MT_HIGH_VECTORS;
24. #else
25.         map.type = MT_LOW_VECTORS;
26. #endif
27.         create_mapping(&map);
28.
29.         if (!vectors_high()) {
30.                 map.virtual = 0;                      ④
31.                 map.length = PAGE_SIZE * 2;
```

```
32.                 map.type = MT_LOW_VECTORS;
33.                 create_mapping(&map);
34.             }
35.
36.         ......
37.
38.     }
```

①

分配 2 physical pages, 第一页是为存放vector table的。

②

在分配的physical page上初始化vector table

③

把初始化了的vector table的physical page mapping to 0xffff0000

④

如果vector table不在high end，则同时把该physical page mapping to 0

```
1.  void __init early_trap_init(void *vectors_base)
2.  {
3.  #ifndef CONFIG_CPU_V7M
4.          unsigned long vectors = (unsigned long)vectors_base;
5.          extern char __stubs_start[], __stubs_end[];
6.          extern char __vectors_start[], __vectors_end[];
7.          unsigned i;
8.
9.          vectors_page = vectors_base;
10.
11.         /*
12.          * Poison the vectors page with an undefined instruction.  This
13.          * instruction is chosen to be undefined for both ARM and Thumb
14.          * ISAs.  The Thumb version is an undefined instruction with a
15.          * branch back to the undefined instruction.
16.          */
17.         for (i = 0; i < PAGE_SIZE / sizeof(u32); i++)
18.                 ((u32 *)vectors_base)[i] = 0xe7fddef1;
19.
20.         /*
21.          * Copy the vectors, stubs and kuser helpers (in entry-armv.S)
22.          * into the vector page, mapped at 0xffff0000, and ensure these
23.          * are visible to the instruction stream.
24.          */
25. memcpy((void *)vectors,  __vectors_start, __vectors_end - __vectors_start);①
26.         memcpy((void *)vectors + 0x1000, __stubs_start, __stubs_end - __stubs_sta
    rt);
27.
28.         kuser_init(vectors_base);
29.
30.         flush_icache_range(vectors, vectors + PAGE_SIZE * 2);
```

```
31.          modify_domain(DOMAIN_USER, DOMAIN_CLIENT);
32.  #else /* ifndef CONFIG_CPU_V7M */
33.          /*
34.           * on V7-M there is no need to copy the vector table to a dedicated
35.           * memory area. The address is configurable and so a table in the kernel
36.           * image can be used.
37.           */
38.  #endif
39.  }
```

①

初始化vector table。

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

```
1.          .section .vectors, "ax", %progbits
2.  __vectors_start:
3.          W(b)    vector_rst
4.          W(b)    vector_und
5.          W(ldr)  pc, __vectors_start + 0x1000
6.          W(b)    vector_pabt
7.          W(b)    vector_dabt
8.          W(b)    vector_addrexcptn
9.          W(b)    vector_irq
10.         W(b)    vector_fiq
```

__vectors_end is in vmlinux.lds

```
/*
        * The vectors and stubs are relocatable code, and the
        * only thing that matters is their relative offsets
        */
__vectors_start = .;
.vectors 0 : AT(__vectors_start) {
 *(.vectors)
}
. = __vectors_start + SIZEOF(.vectors);
__vectors_end = .;
```

```
Disassembly of section .vectors:

00000000 <__vectors_start>:
   0:   ea0003ff        b       1004 <vector_rst>
   4:   ea000465        b       11a0 <vector_und>
   8:   e59ffff0        ldr     pc, [pc, #4080] ; 1000 <__stubs_start>
   c:   ea000443        b       1120 <vector_pabt>
  10:   ea000422        b       10a0 <vector_dabt>
  14:   ea000481        b       1220 <vector_addrexcptn>
  18:   ea000400        b       1020 <vector_irq>
  1c:   ea000487        b       1240 <vector_fiq_offset>
```

在上面allocate的2 physical pages的第一页的头上就是这里的vector table。

vector table中的function address都指向2 physical pages中的第二页,即section .stubs 中的code。从对vmlinux的反汇编中反而比源码(entry-armv.S)更能看懂这点。由于在source code中是通过macro实现的，所以反而unreadable。

```
Disassembly of section .stubs:

00001000 <__stubs_start>:
    1000:       c000e760        .word   0xc000e760

00001004 <vector_rst>:
    1004:       ef9f0000        svc     0x009f0000
    1008:       ea000064        b       11a0 <vector_und>
    100c:       e320f000        nop     {0}
    1010:       e320f000        nop     {0}
    1014:       e320f000        nop     {0}
    1018:       e320f000        nop     {0}
    101c:       e320f000        nop     {0}

00001020 <vector_irq>:
    1020:       e24ee004        sub     lr, lr, #4
    1024:       e88d4001        stm     sp, {r0, lr}
    1028:       e14fe000        mrs     lr, SPSR
    102c:       e58de008        str     lr, [sp, #8]
    1030:       e10f0000        mrs     r0, CPSR
    1034:       e2200001        eor     r0, r0, #1
    1038:       e16ff000        msr     SPSR_fsxc, r0
    103c:       e20ee00f        and     lr, lr, #15
    1040:       e1a0000d        mov     r0, sp
    1044:       e79fe10e        ldr     lr, [pc, lr, lsl #2]
    1048:       e1b0f00e        movs    pc, lr
    104c:       c00122c0        .word   0xc00122c0
    1050:       c0011f60        .word   0xc0011f60
    1054:       c0011f60        .word   0xc0011f60
    1058:       c0012000        .word   0xc0012000
    105c:       c0011f60        .word   0xc0011f60
    1060:       c0011f60        .word   0xc0011f60
    1064:       c0011f60        .word   0xc0011f60
    1068:       c0011f60        .word   0xc0011f60
    106c:       c0011f60        .word   0xc0011f60
    1070:       c0011f60        .word   0xc0011f60
    1074:       c0011f60        .word   0xc0011f60
    1078:       c0011f60        .word   0xc0011f60
    107c:       c0011f60        .word   0xc0011f60
    1080:       c0011f60        .word   0xc0011f60
    1084:       c0011f60        .word   0xc0011f60
    1088:       c0011f60        .word   0xc0011f60
    108c:       e320f000        nop     {0}
    1090:       e320f000        nop     {0}
    1094:       e320f000        nop     {0}
    1098:       e320f000        nop     {0}
    109c:       e320f000        nop     {0}

000010a0 <vector_dabt>:
    10a0:       e24ee008        sub     lr, lr, #8
    10a4:       e88d4001        stm     sp, {r0, lr}
    10a8:       e14fe000        mrs     lr, SPSR
    10ac:       e58de008        str     lr, [sp, #8]
```

```
      10b0:       e10f0000        mrs     r0, CPSR
      10b4:       e2200004        eor     r0, r0, #4
      10b8:       e16ff000        msr     SPSR_fsxc, r0
      10bc:       e20ee00f        and     lr, lr, #15
      10c0:       e1a0000d        mov     r0, sp
      10c4:       e79fe10e        ldr     lr, [pc, lr, lsl #2]
      10c8:       e1b0f00e        movs    pc, lr
      10cc:       c0012280        .word   0xc0012280
      10d0:       c0011f50        .word   0xc0011f50
      10d4:       c0011f50        .word   0xc0011f50
      10d8:       c0011fa0        .word   0xc0011fa0
      10dc:       c0011f50        .word   0xc0011f50
      10e0:       c0011f50        .word   0xc0011f50
      10e4:       c0011f50        .word   0xc0011f50
      10e8:       c0011f50        .word   0xc0011f50
      10ec:       c0011f50        .word   0xc0011f50
      10f0:       c0011f50        .word   0xc0011f50
      10f4:       c0011f50        .word   0xc0011f50
      10f8:       c0011f50        .word   0xc0011f50
      10fc:       c0011f50        .word   0xc0011f50
      1100:       c0011f50        .word   0xc0011f50
      1104:       c0011f50        .word   0xc0011f50
      1108:       c0011f50        .word   0xc0011f50
      110c:       e320f000        nop     {0}
      1110:       e320f000        nop     {0}
      1114:       e320f000        nop     {0}
      1118:       e320f000        nop     {0}
      111c:       e320f000        nop     {0}

00001120 <vector_pabt>:
      1120:       e24ee004        sub     lr, lr, #4
      1124:       e88d4001        stm     sp, {r0, lr}
      1128:       e14fe000        mrs     lr, SPSR
      112c:       e58de008        str     lr, [sp, #8]
      1130:       e10f0000        mrs     r0, CPSR
      1134:       e2200004        eor     r0, r0, #4
      1138:       e16ff000        msr     SPSR_fsxc, r0
      113c:       e20ee00f        and     lr, lr, #15
      1140:       e1a0000d        mov     r0, sp
      1144:       e79fe10e        ldr     lr, [pc, lr, lsl #2]
      1148:       e1b0f00e        movs    pc, lr
      114c:       c00124e0        .word   0xc00124e0
      1150:       c0011f40        .word   0xc0011f40
      1154:       c0011f40        .word   0xc0011f40
      1158:       c0012120        .word   0xc0012120
      115c:       c0011f40        .word   0xc0011f40
      1160:       c0011f40        .word   0xc0011f40
      1164:       c0011f40        .word   0xc0011f40
      1168:       c0011f40        .word   0xc0011f40
      116c:       c0011f40        .word   0xc0011f40
      1170:       c0011f40        .word   0xc0011f40
      1174:       c0011f40        .word   0xc0011f40
      1178:       c0011f40        .word   0xc0011f40
      117c:       c0011f40        .word   0xc0011f40
```

```
    1180:       c0011f40        .word   0xc0011f40
    1184:       c0011f40        .word   0xc0011f40
    1188:       c0011f40        .word   0xc0011f40
    118c:       e320f000        nop     {0}
    1190:       e320f000        nop     {0}
    1194:       e320f000        nop     {0}
    1198:       e320f000        nop     {0}
    119c:       e320f000        nop     {0}

000011a0 <vector_und>:
    11a0:       e88d4001        stm     sp, {r0, lr}
    11a4:       e14fe000        mrs     lr, SPSR
    11a8:       e58de008        str     lr, [sp, #8]
    11ac:       e10f0000        mrs     r0, CPSR
    11b0:       e2200008        eor     r0, r0, #8
    11b4:       e16ff000        msr     SPSR_fsxc, r0
    11b8:       e20ee00f        and     lr, lr, #15
    11bc:       e1a0000d        mov     r0, sp
    11c0:       e79fe10e        ldr     lr, [pc, lr, lsl #2]
    11c4:       e1b0f00e        movs    pc, lr
    11c8:       c0012320        .word   0xc0012320
    11cc:       c0011f70        .word   0xc0011f70
    11d0:       c0011f70        .word   0xc0011f70
    11d4:       c00120a0        .word   0xc00120a0
    11d8:       c0011f70        .word   0xc0011f70
    11dc:       c0011f70        .word   0xc0011f70
    11e0:       c0011f70        .word   0xc0011f70
    11e4:       c0011f70        .word   0xc0011f70
    11e8:       c0011f70        .word   0xc0011f70
    11ec:       c0011f70        .word   0xc0011f70
    11f0:       c0011f70        .word   0xc0011f70
    11f4:       c0011f70        .word   0xc0011f70
    11f8:       c0011f70        .word   0xc0011f70
    11fc:       c0011f70        .word   0xc0011f70
    1200:       c0011f70        .word   0xc0011f70
    1204:       c0011f70        .word   0xc0011f70
    1208:       e320f000        nop     {0}
    120c:       e320f000        nop     {0}
    1210:       e320f000        nop     {0}
    1214:       e320f000        nop     {0}
    1218:       e320f000        nop     {0}
    121c:       e320f000        nop     {0}

00001220 <vector_addrexcptn>:
    1220:       eafffffe        b       1220 <vector_addrexcptn>
    1224:       e320f000        nop     {0}
    1228:       e320f000        nop     {0}
    122c:       e320f000        nop     {0}
    1230:       e320f000        nop     {0}
    1234:       e320f000        nop     {0}
    1238:       e320f000        nop     {0}
    123c:       e320f000        nop     {0}

00001240 <vector_fiq_offset>:
```

```
      1240:       e24ee004        sub     lr, lr, #4
      1244:       e88d4001        stm     sp, {r0, lr}
      1248:       e14fe000        mrs     lr, SPSR
      124c:       e58de008        str     lr, [sp, #8]
      1250:       e10f0000        mrs     r0, CPSR
      1254:       e2200002        eor     r0, r0, #2
      1258:       e16ff000        msr     SPSR_fsxc, r0
      125c:       e20ee00f        and     lr, lr, #15
      1260:       e1a0000d        mov     r0, sp
      1264:       e79fe10e        ldr     lr, [pc, lr, lsl #2]
      1268:       e1b0f00e        movs    pc, lr
      126c:       c0012540        .word   0xc0012540
      1270:       c0012180        .word   0xc0012180
      1274:       c0012180        .word   0xc0012180
      1278:       c0012180        .word   0xc0012180
      127c:       c0012180        .word   0xc0012180
      1280:       c0012180        .word   0xc0012180
      1284:       c0012180        .word   0xc0012180
      1288:       c0012200        .word   0xc0012200
      128c:       c0012180        .word   0xc0012180
      1290:       c0012180        .word   0xc0012180
      1294:       c0012180        .word   0xc0012180
      1298:       c0012180        .word   0xc0012180
      129c:       c0012180        .word   0xc0012180
      12a0:       c0012180        .word   0xc0012180
      12a4:       c0012180        .word   0xc0012180
      12a8:       c0012180        .word   0xc0012180
      12ac:       e320f000        nop     {0}
      12b0:       e320f000        nop     {0}
      12b4:       e320f000        nop     {0}
      12b8:       e320f000        nop     {0}
      12bc:       e320f000        nop     {0}
```

以vector_dabt()为例

源码为

```
/*
 * Data abort dispatcher
 * Enter in ABT mode, spsr = USR CPSR, lr = USR PC
 */
        vector_stub     dabt, ABT_MODE, 8

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

vector_stub is macro

反汇编code为

000010a0 <vector_dabt>:

10a0: e24ee008 sub lr, lr, #8 ①

10a4: e88d4001 stm sp, {r0, lr}

10a8: e14fe000 mrs lr, SPSR

10ac: e58de008 str lr, [sp, #8]

10b0: e10f0000 mrs r0, CPSR

10b4: e2200004 eor r0, r0, #4

10b8: e16ff000 msr SPSR_fsxc, r0

```
10bc:   e20ee00f        and  lr, lr, #15
10c0:   e1a0000d        mov r0, sp
10c4:   e79fe10e        ldr    lr, [pc, lr, lsl #2]
10c8:   e1b0f00e        movs       pc, lr

----------------------------------------

10cc:   c0012280        .word      0xc0012280          ②
10d0:   c0011f50        .word      0xc0011f50
10d4:   c0011f50        .word      0xc0011f50
10d8:   c0011fa0        .word      0xc0011fa0
10dc:   c0011f50        .word      0xc0011f50
10e0:   c0011f50        .word      0xc0011f50
10e4:   c0011f50        .word      0xc0011f50
10e8:   c0011f50        .word      0xc0011f50
10ec:   c0011f50        .word      0xc0011f50
10f0:   c0011f50        .word      0xc0011f50
10f4:   c0011f50        .word      0xc0011f50
10f8:   c0011f50        .word      0xc0011f50
10fc:   c0011f50        .word      0xc0011f50
1100:   c0011f50        .word      0xc0011f50
1104:   c0011f50        .word      0xc0011f50
1108:   c0011f50        .word      0xc0011f50
```

汇编码中的①对应source code中的

vector_stub dabt, ABT_MODE, 8

汇编码中的②是不同mode下的data abort的handler。由core当前运行的mode作为index来跳转到对应的handler。由于Linux中工作在2种mode(kernel 运行在SVC mode,而application运行在user mode)，所以table中真正有用的是__dabt_usr()和__dabt_svc()。即当在application中出现data abort，则跳转到__dabt_usr()；而kernel中出现则跳转到__dabt_svc()。

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

```
        .align  5
__dabt_usr:
        usr_entry
        kuser_cmpxchg_check
        mov     r2, sp
        dabt_helper
        b       ret_from_exception
 UNWIND(.fnend          )
ENDPROC(__dabt_usr)

        .align  5
__dabt_svc:
        svc_entry
        mov     r2, sp
        dabt_helper                 ①
 THUMB( ldr     r5, [sp, #S_PSR]        )       @ potentially updated CPSR
        svc_exit r5                             @ return from exception
 UNWIND(.fnend          )
ENDPROC(__dabt_svc)
```

①

```
        .macro  dabt_helper

        @
        @ Call the processor-specific abort handler:
        @
        @  r2 - pt_regs
        @  r4 - aborted context pc
        @  r5 - aborted context psr
        @
        @ The abort handler must return the aborted address in r0, and
        @ the fault status register in r1.  r9 must be preserved.
        @
#ifdef MULTI_DABORT
        ldr     ip, .LCprocfns
        mov     lr, pc
        ldr     pc, [ip, #PROCESSOR_DABT_FUNC]
#else
        bl      CPU_DABORT_HANDLER          ②
#endif
        .endm
```

②

这是个因ARMv?不同而不同的macro。

in arch/arm/include/asm/glue-df.h

对Granite2 / Gemstone2而言

```
#ifdef CONFIG_CPU_ABRT_EV7
# ifdef CPU_DABORT_HANDLER
#  define MULTI_DABORT 1
# else
#  define CPU_DABORT_HANDLER v7_early_abort
# endif
#endif
```

即bl CPU_DABORT_HANDLER

==>

bl v7_early_abort

---

in arch/arm/mm/abort-ev7.S

```
/*
 * Function: v7_early_abort
 *
 * Params  : r2 = pt_regs
 *         : r4 = aborted context pc
 *         : r5 = aborted context psr
 *
 * Returns : r4 - r11, r13 preserved
 *
 * Purpose : obtain information about current aborted instruction.
 */
        .align  5
ENTRY(v7_early_abort)
        mrc     p15, 0, r1, c5, c0, 0           @ get FSR
        mrc     p15, 0, r0, c6, c0, 0           @ get FAR

        /*
         * V6 code adjusts the returned DFSR.
         * New designs should not need to patch up faults.
         */

#if defined(CONFIG_VERIFY_PERMISSION_FAULT)
        /*
         * Detect erroneous permission failures and fix
         */
        ldr     r3, =0x40d                      @ On permission fault
        and     r3, r1, r3
        cmp     r3, #0x0d
        bne     do_DataAbort

        mcr     p15, 0, r0, c7, c8, 0           @ Retranslate FAR
        isb
        mrc     p15, 0, ip, c7, c4, 0           @ Read the PAR
        and     r3, ip, #0x7b                   @ On translation fault
        cmp     r3, #0x0b
        bne     do_DataAbort
        bic     r1, r1, #0xf                    @ Fix up FSR FS[5:0]
        and     ip, ip, #0x7e
        orr     r1, r1, ip, LSR #1
#endif

        b       do_DataAbort
ENDPROC(v7_early_abort)
```

由于在G2 LSP中，CONFIG_VERIFY_PERMISSION_FAULT并没有enable，所以v7_early_abort()

其实就是如下code：

```
ENTRY(v7_early_abort)
        mrc     p15, 0, r1, c5, c0, 0           @ get FSR
        mrc     p15, 0, r0, c6, c0, 0           @ get FAR

/*
         * V6 code adjusts the returned DFSR.
         * New designs should not need to patch up faults.
         */

        b       do_DataAbort
ENDPROC(v7_early_abort)
```

r0 --> unsigned long addr

r1 --> unsigned int fsr

r2 --> pt_regs

in arch/arm/mm/fault.c

```
1.  /*
2.   * Dispatch a data abort to the relevant handler.
3.   */
4.  asmlinkage void __exception
5.  do_DataAbort(unsigned long addr, unsigned int fsr, struct pt_regs *regs)
6.  {
7.          const struct fsr_info *inf =fsr_info+ fsr_fs(fsr);        ①
8.          struct siginfo info;
9.
10.         if ((fsr == 0xc06) || (fsr == 0xa11)) {
11.                 printk(KERN_EMERG "FIX ignoring exception %#x addr=%lx %s:%d\n\n"
    , fsr, addr, current->comm, current->pid);
12.                 return;
13.         }
14.
15.         if (!inf->fn(addr, fsr & ~FSR_LNX_PF, regs))
16.                 return;
17.
18.         printk(KERN_ALERT "Unhandled fault: %s (0x%03x) at 0x%08lx\n",
19.                 inf->name, fsr, addr);
20.
21.         info.si_signo = inf->sig;
22.         info.si_errno = 0;
23.         info.si_code  = inf->code;
24.         info.si_addr  = (void __user *)addr;
25.         arm_notify_die("", regs, &info, fsr, 0);
26. }
```

①

fsr_info array中存放了不同fault status下的不同handler。

in arch/arm/mm/fsr-2level.c

```
static struct fsr_info fsr_info[] = {
        /*
         * The following are the standard ARMv3 and ARMv4 aborts.  ARMv5
         * defines these to be "precise" aborts.
         */
        { do_bad,               SIGSEGV, 0,             "vector exception"
           },
        { do_bad,               SIGBUS,  BUS_ADRALN,    "alignment exception"
           },
        { do_bad,               SIGKILL, 0,             "terminal exception"
           },
        { do_bad,               SIGBUS,  BUS_ADRALN,    "alignment exception"
           },
        { do_bad,               SIGBUS,  0,             "external abort on linefe
tch"       },
        { do_translation_fault, SIGSEGV, SEGV_MAPERR,   "section translation faul
t"         },
        { do_bad,               SIGBUS,  0,             "external abort on linefe
tch"       },
        { do_page_fault,        SIGSEGV, SEGV_MAPERR,   "page translation fault"
           },
        { do_bad,               SIGBUS,  0,             "external abort on non-li
nefetch"   },
        { do_bad,               SIGSEGV, SEGV_ACCERR,   "section domain fault"
           },
        { do_bad,               SIGBUS,  0,             "external abort on non-li
nefetch"   },
        { do_bad,               SIGSEGV, SEGV_ACCERR,   "page domain fault"
           },
        { do_bad,               SIGBUS,  0,             "external abort on transl
ation"     },
        { do_sect_fault,        SIGSEGV, SEGV_ACCERR,   "section permission fault
"          },
        { do_bad,               SIGBUS,  0,             "external abort on transl
ation"     },
        { do_page_fault,        SIGSEGV, SEGV_ACCERR,   "page permission fault"
           },
        /*
         * The following are "imprecise" aborts, which are signalled by bit
         * 10 of the FSR, and may not be recoverable.  These are only
         * supported if the CPU abort handler supports bit 10.
         */
        { do_bad,               SIGBUS,  0,             "unknown 16"
           },
        { do_bad,               SIGBUS,  0,             "unknown 17"
           },
        { do_bad,               SIGBUS,  0,             "unknown 18"
           },
        { do_bad,               SIGBUS,  0,             "unknown 19"
           },
        { do_bad,               SIGBUS,  0,             "lock abort"
           }, /* xscale */
        { do_bad,               SIGBUS,  0,             "unknown 21"
```

```
            },
        { do_bad,               SIGBUS,  BUS_OBJERR,    "imprecise external abort
"           }, /* xscale */
        { do_bad,               SIGBUS,  0,             "unknown 23"
            },
        { do_bad,               SIGBUS,  0,             "dcache parity error"
            }, /* xscale */
        { do_bad,               SIGBUS,  0,             "unknown 25"
            },
        { do_bad,               SIGBUS,  0,             "unknown 26"
            },
        { do_bad,               SIGBUS,  0,             "unknown 27"
            },
        { do_bad,               SIGBUS,  0,             "unknown 28"
            },
        { do_bad,               SIGBUS,  0,             "unknown 29"
            },
        { do_bad,               SIGBUS,  0,             "unknown 30"
            },
        { do_bad,               SIGBUS,  0,             "unknown 31"
            },
};
```

比如由application generate data abort而引起的page fault的处理，那么整个call down chain应该是这样的：

1. vector table中的第4 entry(zero-based), vector_dabt in entry-armv.S

2. vector_stub dabt in entry-armv.S

3. __dabt_usr() in entry-armv.S

4. dabt_helper macro in entry-armv.S

5. v7_early_abort() in abort-ev7.S

6. do_DataAbort() in arch/arm/mm/fault.c

7. do_page_fault() in arch/arm/mm/fault.c