- modify arch/arm/Makefile

```
walterzh@walterzh-Precision-T1650:~/work/current/tmp/work-shared/granite2/kernel-source/arch/arm$ diff Makefile ~/work2/original-kernel-src/linux-3.18.7/arch/arm/Makefile
121,122c121,122
< KBUILD_CFLAGS +=$(CFLAGS_ABI) $(CFLAGS_ISA) $(arch-y) $(tune-y) $(call cc-option,-mshort-load-bytes,$(call cc-option,-malignment-traps,)) -mhard-float -Uarm
< KBUILD_AFLAGS +=$(CFLAGS_ABI) $(AFLAGS_ISA) $(arch-y) $(tune-y) -include asm/unified.h -mhard-float
---
> KBUILD_CFLAGS +=$(CFLAGS_ABI) $(CFLAGS_ISA) $(arch-y) $(tune-y) $(call cc-option,-mshort-load-bytes,$(call cc-option,-malignment-traps,)) -msoft-float -Uarm
> KBUILD_AFLAGS +=$(CFLAGS_ABI) $(AFLAGS_ISA) $(arch-y) $(tune-y) -include asm/unified.h -msoft-float
```

convert `-msoft-float option` into `-mhard-float`

- enable kernel support VFP (NEON)
  CONFIG_KERNEL_MODE_NEON=y


- How to do floating point computing in kernel code

```c
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/init.h>
#include <asm/neon.h>    ①

static int __init test_module_init(void)
{
    float d1, d2;

    kernel_neon_begin();    ②

    d1 = 1234.5;
    d2 = d1 / 6;

    printk("1234.5 / 6 = %d\n", (int)d2);    ③
    kernel_neon_end();    ④

    return 0;
}

static void __exit test_module_exit(void)
{
}
module_init(test_module_init);
module_exit(test_module_exit);
MODULE_LICENSE("GPL");
```

```
walterzh@walterzh-Precision-T1650:~/work/victor/module-test/test$ cat Makefile
#
# ============================================================================
# Copyright (c) 2015 Marvell International, Ltd. All Rights Reserved
#
# Marvell Confidential
# ============================================================================
#

obj-m := test.o

ccflags-y += -O0 -march=armv7-a -mhard-float    ⑤

SRC := $(shell pwd)

all:
    echo "test build"
    echo $(KERNEL_SRC)
    echo $(ccflags-y)
    $(MAKE) -C $(KERNEL_SRC) M=$(SRC)
modules_install:
    echo "install test"
    echo $(KERNEL_SRC)
    $(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules_install

clean:
    rm -f *.o *~ core .depend .*.cmd *.ko *.mod.c
    rm -f Module.markers Module.symvers modules.order
    rm -rf .tmp_versions Modules.symvers
```

①

You need include neon.h for accessing kernel_neon_begin() and kernel_neon_end() APIs

②④

All floating point computing must be embedded in kernel_neon_begin() and kernel_neon_end(),
otherwise kernel will crash.

③

Because printk doesn't support "%f" specifier, so I convert float number into integer number to output.

⑤

You need add `-mhard-float` option to compile module

- internal of floating point computing

```
walterzh@walterzh-Precision-T1650:~/work/victor/module-test/test$ arm-linux-gnueabi-objdump -dS test.ko

test.ko:     file format elf32-littlearm


Disassembly of section .init.text:

00000000 <init_module>:
#include <linux/init.h>
#include <asm/neon.h>
#include <linux/preempt.h>

static int __init test_module_init(void)
{
   0:   e52de004    push    {lr}        ; (str lr, [sp, #-4]!)
   4:   e24dd00c    sub sp, sp, #12
    float d1, d2;

    kernel_neon_begin();
   8:   ebfffffe    bl  0 <kernel_neon_begin>

    d1 = 1234.5;
   c:   e3a03a05    mov r3, #20480  ; 0x5000
  10:   e344349a    movt    r3, #17562  ; 0x449a
  14:   e58d3004    str r3, [sp, #4]
    d2 = d1 / 6;
  18:   eddd7a01    vldr    s15, [sp, #4]        ⑥
  1c:   ed9f7a0c    vldr    s14, [pc, #48]  ; 54 <init_module+0x54>
  20:   eec77a87    vdiv.f32    s15, s15, s14
  24:   edcd7a00    vstr    s15, [sp]

    printk("1234.5 / 6 = %d\n", (int)d2);
  28:   eddd7a00    vldr    s15, [sp]            ⑦
  2c:   eefd7ae7    vcvt.s32.f32    s15, s15
  30:   e3000000    movw    r0, #0
  34:   e3400000    movt    r0, #0
  38:   ee171a90    vmov    r1, s15
  3c:   ebfffffe    bl  0 <printk>
    kernel_neon_end();
  40:   ebfffffe    bl  0 <kernel_neon_end>

    return 0;
  44:   e3a03000    mov r3, #0
}
  48:   e1a00003    mov r0, r3
  4c:   e28dd00c    add sp, sp, #12
  50:   e49df004    pop {pc}        ; (ldr pc, [sp], #4)
  54:   40c00000    .word   0x40c00000

Disassembly of section .exit.text:

00000000 <cleanup_module>:
```

```
53.
54.  static void __exit test_module_exit(void)
55.  {
56.  }
57.     0:   e12fff1e    bx  lr
```

⑥⑦
generate VFP(NEON) instruction

- Running result

```
1.  root@granite2:~# insmod test.ko
2.  1234.5 / 6 = 205
```

- Restriction
  floating point computing in kernel could run in the following context

1. non-interrupt context
2. preemption disable

That means you could not do floating point computing in interrupt service routine and you should not do time-consuming computing in kernel (Because preemption is disable, scheduler has been locked)

- The impact on kernel

Because `-mhard-float` option influenced deeply the generated kernel code, maybe we need more investigation to ensure whether the modification is safe.

for example:

in kernel/time/timeconv.c

```
1.  /* do a mathdiv for long type */
2.  static long math_div(long a, long b)
3.  {
4.      return a / b - (a % b < 0);
5.  }
```

if we use original gcc compiler option `-msoft-float`, the generated code as follow

```
/* do a mathdiv for long type */
static long math_div(long a, long b)
{
    return a / b - (a % b < 0);
c006b994:   e59d3010    ldr r3, [sp, #16]
c006b998:   e0c10396    smull   r0, r1, r6, r3
c006b99c:   e1a03fc6    asr r3, r6, #31
c006b9a0:   e0862001    add r2, r6, r1
c006b9a4:   e0633442    rsb r3, r3, r2, asr #8
c006b9a8:   e0832183    add r2, r3, r3, lsl #3
c006b9ac:   e0832182    add r2, r3, r2, lsl #3
c006b9b0:   e0822102    add r2, r2, r2, lsl #2
c006b9b4:   e0622006    rsb r2, r2, r6
c006b9b8:   e0433fa2    sub r3, r3, r2, lsr #31
```

Because the code has been optimized by `-02`, it's not readable. But we could identify gcc doesn't generate VFP(NEON) instruction to implement "div" operation.

if we use gcc compiler option `-mhard-float`, the generated code as follow

```
c0071fec <math_div>:

/* do a mathdiv for long type */
static long math_div(long a, long b)
{
c0071fec:   e92d4070    push    {r4, r5, r6, lr}
c0071ff0:   e1a06000    mov r6, r0
c0071ff4:   e1a05001    mov r5, r1
    return a / b - (a % b < 0);
c0071ff8:   eb0ab4bb    bl  c031f2ec <__aeabi_idiv>
c0071ffc:   e1a04000    mov r4, r0
c0072000:   e1a00006    mov r0, r6
c0072004:   e1a01005    mov r1, r5
c0072008:   eb0ab51e    bl  c031f488 <__aeabi_idivmod>
}
c007200c:   e0440fa1    sub r0, r4, r1, lsr #31
c0072010:   e8bd8070    pop {r4, r5, r6, pc}
```

`__aeabi_idiv` and `__aeabi_idivmod` are introduced by "-mhard-float" option.

```
walterzh@walterzh-Precision-T1650:~/work/current/work/granite2-poky-linux-gnueabi/linux-granite-upstream/3.18.7+gitAUTOINC+26304af6aa-r0/image/boot$ arm-linux-gnueabi-nm vmlinux-3.18.7-yocto-standard | grep " __aeabi_"
c031f2ec T __aeabi_idiv
c031f488 T __aeabi_idivmod
c031d52c T __aeabi_lasr
c031d510 T __aeabi_llsl
c031f4b0 T __aeabi_llsr
c031fce0 T __aeabi_lmul
c031f1b0 T __aeabi_uidiv
c031f470 T __aeabi_uidivmod
c031ff3c T __aeabi_ulcmp
c0015ce4 T __aeabi_unwind_cpp_pr0
c0015ce8 T __aeabi_unwind_cpp_pr1
c0015cec T __aeabi_unwind_cpp_pr2
```

These functions are from libgcc.a.

**Amend:**

We need not rebuild kernel by replacing `-mhard-float` option with `-msoft-float`.
The kernel is still VFP instruction free, built by `-msoft-float` option, but when build out-of-tree kernel module, we need modified arch/arm/Makefile.

**So, do floating point computing in out-of-tree kernel module code is safe.**