T-HEAD

Kprobes Jump Optimized *OPTPROBES* for more Archs

Guo Ren
<guoren@kernel.org>
<guoren@linux.alibaba.com>

Self Introduction

- Focus on T-HEAD Xuantie CPUs' linux porting
  - RV64GCV:
    - C910: 12-stage pipeline, 3-issue, 8-execution, SMP with multi-clusters
  - C-SKY 32bit:
    - C807: 8-stage pipeline, 2-issue, low power
    - C810: 10-stage pipeline, 2-issue, 5-execution
    - C860: 12-stage pipeline, 3-issue, 8-execution, SMP

More details ref to t-head

- Linux/arch/csky subsystem maintainer

My recent work

- Author of csky's ftrace, k/uprobe
  - 8f6bb79 csky: Add uprobes support
  - 33e53ae csky: Add kprobes supported
  - 28bb030 csky/ftrace: Add dynamic function tracer with call-graph
  - 89a3927 csky: Implement ftrace with regs
  - d7950be csky: ftrace call graph supported
  - 230c77a csky: basic ftrace supported
- The one of contributors for riscv k/uprobe
  - [PATCH v3 4/7] riscv: Add kprobes supported
  - [PATCH v3 5/7] riscv: Add uprobes supported
  - [PATCH v3 6/7] riscv: Add KPROBES_ON_FTRACE supported
  - [PATCH v3 7/7] riscv: Add support for function error injection

Statistics in linux-5.8

- Archs with K&UPROBES:
  (powerpc, arm32/64, s390, sparc, x86, parisc, csky, mips)

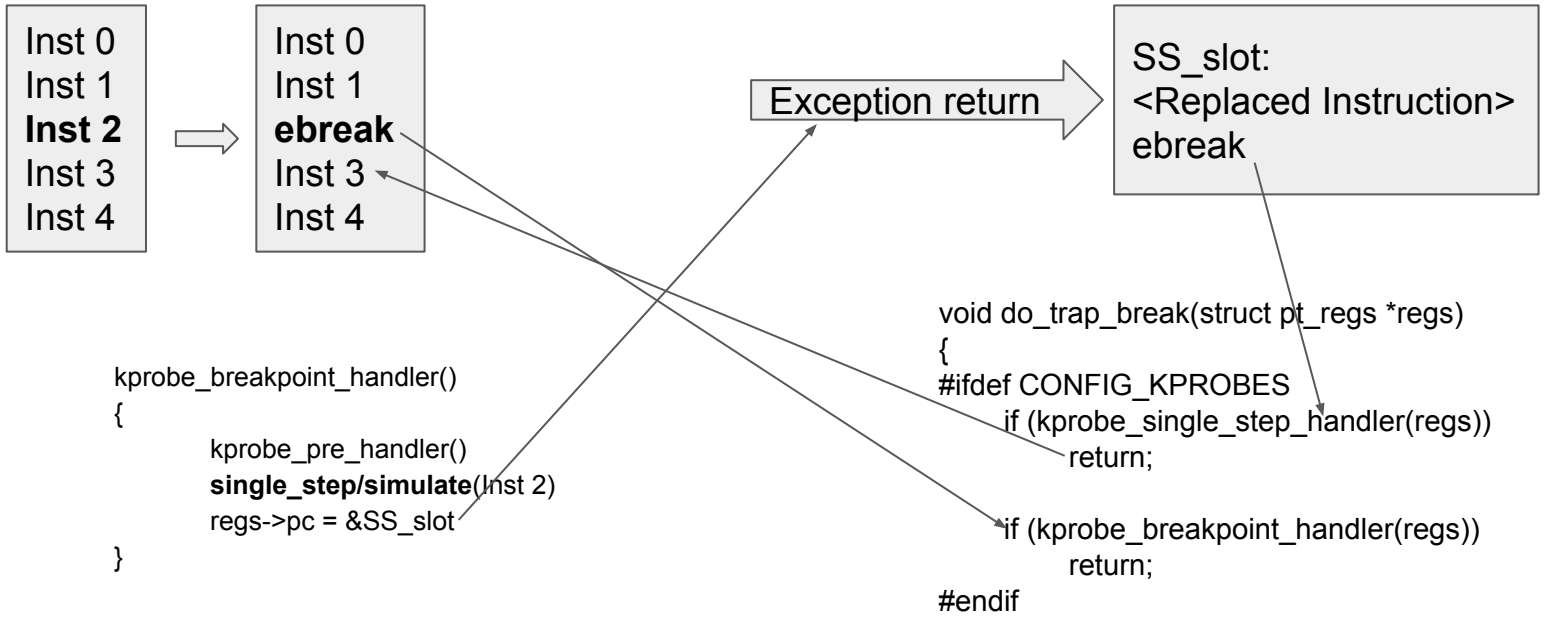
- Archs with LIVEPATCH:
  (powerpc, s390, x86)

- ***Archs with OPTPROBES:***
  (powerpc, arm32, x86)

GOAL: Let RISC-V & C-SKY support the features above this year

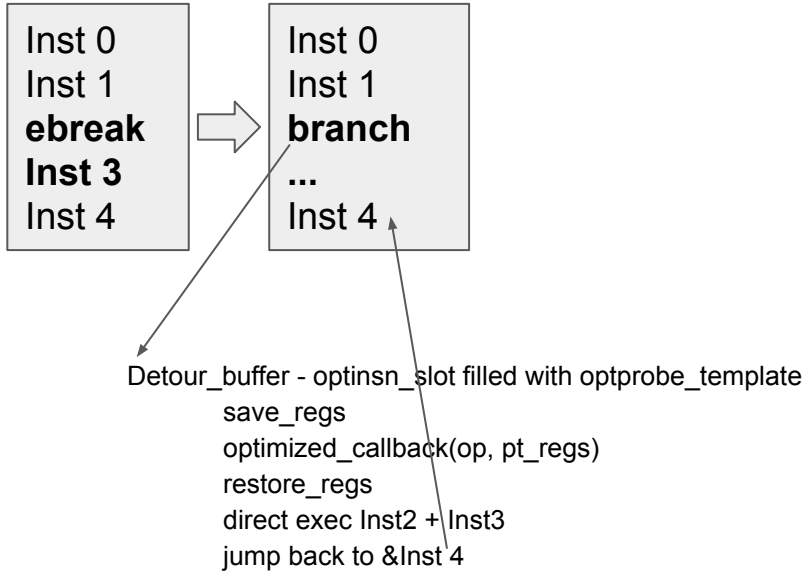
Performance benefit

k = unoptimized kprobe, b = boosted (single-step skipped), o = optimized kprobe
x86-64: Intel(R) Xeon(R) E5410, 2.33GHz, 4656.90 bogomips
k = 0.99 usec; b = 0.43; ***o = 0.06***;



Kprobes Jump Optimization

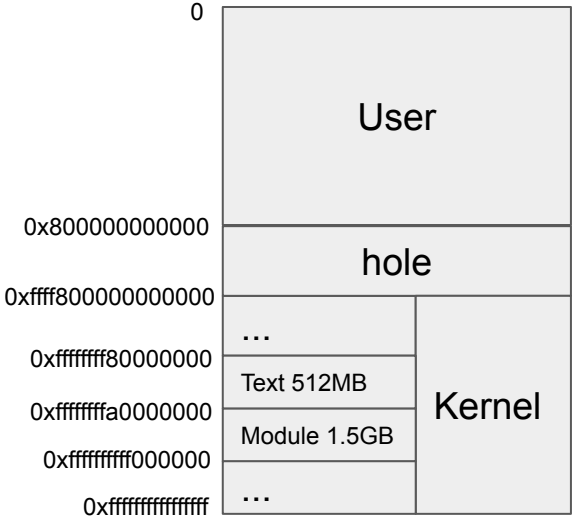
- Pre:
  - Setup one detour buffer to the opt-kprobe (**1 -> 1**)
  - Replace the breakpoint with a branch instruction
- Hit:
  - Branch to detour buffer
  - Save regs
  - Optimized_callback (No post-handler)
  - restore regs
  - exec replaced instructions
  - Jump back to the original execution path



X86 vs. Arm32 vs. Powerpc

X86

- Using a 5 bytes branch instruction with 2GB range
- **All the replaced instructions must be:**
  - **Relocatable**
  - **Not include a call instruction**
- **Couldn’t reuse kprobe single-step skipped, because many instructions**

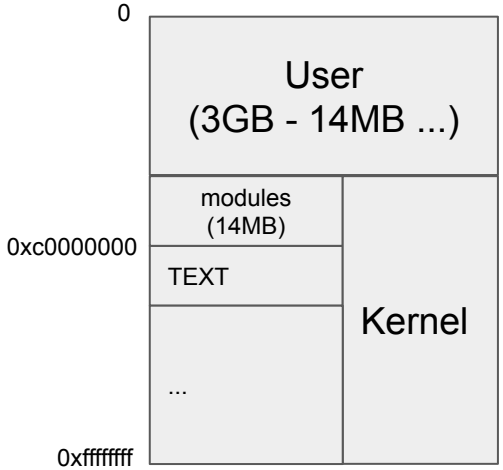


x86_64 mm layout

Arm32

- All instructions are 4 bytes
- Using branch instruction with 32MB range
- Only one instruction was replaced
- Support kprobe single-step skipped
  - Some non-relocatable instruction could be simulated
  - It couldn't change return pc, because detour final branch has been generated

B/BL support (-128M, 128M) offset ARM64 virtual address arrangement guarantees all kernel and module texts are within +/-128M. Why no arm64? Barry test BCC's funclatency: https://www.spinics.net/lists/arm-kernel/msg828788.html



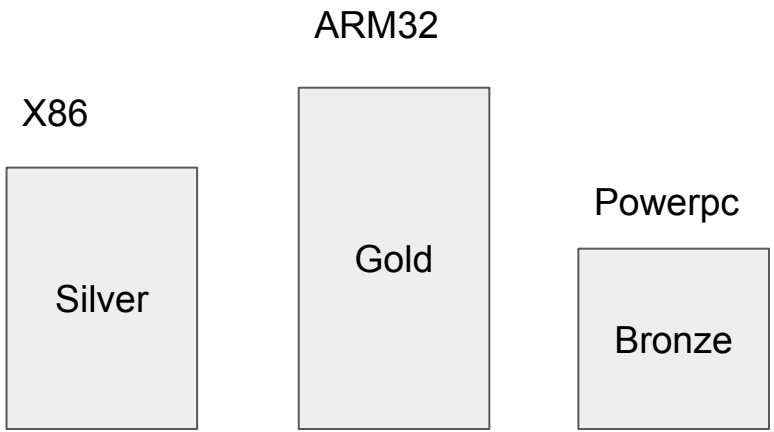
Arm32 mm layout

Powerpc

- All instructions are 4 bytes
- Using branch instruction with 32MB range
- Only one instruction was replaced
- **Only one optinsn_slot**
  (Only one optprobe could be enabled once)
- **Can’t cross module text**
- **Not support kprobe single-step skipped (forgot?)**
  - Relocatable
  - Not include a call instruction

```
arch/powerpc/kernel/optprobes_head.S:
#define OPT_SLOT_SIZE   65536

        .balign 4

        /*
         * Reserve an area to allocate slots for detour buffer.
         * This is part of .text section (rather than vmalloc area)
         * as this needs to be within 32MB of the probed address.
         */
        .global optinsn_slot
optinsn_slot:
        .space  OPT_SLOT_SIZE

arch/powerpc/kernel/optprobes.c:
static void *__ppc_alloc_insn_page(void)
{
        if (insn_page_in_use)
                return NULL;
        insn_page_in_use = true;
        return &optinsn_slot;

kernel/kprobes.c:
void __weak *alloc_insn_page(void)
{
        return module_alloc(PAGE_SIZE);
}
```

Awards



Puzzles of RISC-V & C-SKY

RV64GC:

- J offset[20 bits wide] with +/- 512KB range
- 16/32bits mix opcode, similar to x86
- .text is far from .modules

C-SKY:

- J offset[16 bits wide] with +/- 32KB range
- 16/32bits mix opcode, similar to x86
- .text is far from .modules

No proper Jump Instruction to use

Solution for rv64

Solution 1:

- Reserve some clobber registers, and use one to jump (similar to arm64 x9-x17)
- **auipc x9, offset_20bit**
  jr offset_12bit(x9)
  Total cost is 8 bytes, we got +/- 2GB range. Similar to arm64
- Redesign module's memory layout close to .text in 2GB range, similar to x86

Solution 2:

- Reserve some clobber registers, and use one here to jump (similar to arm64 x9-x17)
- la x9, symbol
  jr 0(x9)
  Total cost is >8 bytes which could use the whole range of **64-bit**
  'la' is pseudo instruction = auipc + addi ...

Solution for csky32

Ref to arch/csky/kernel/ftrace.c:

- r26 is reserved to clobber in csky abi
- movih r26, imm
  ori r26, imm
  jsr r26
  Total cost is 12 bytes and similar to x86 5 bytes jump instruction, we got +/- 2GB range. It’s enough to 32-bit machines.

An ISA Idea for OPTPROBES

Shadow Program Counter (SPC)

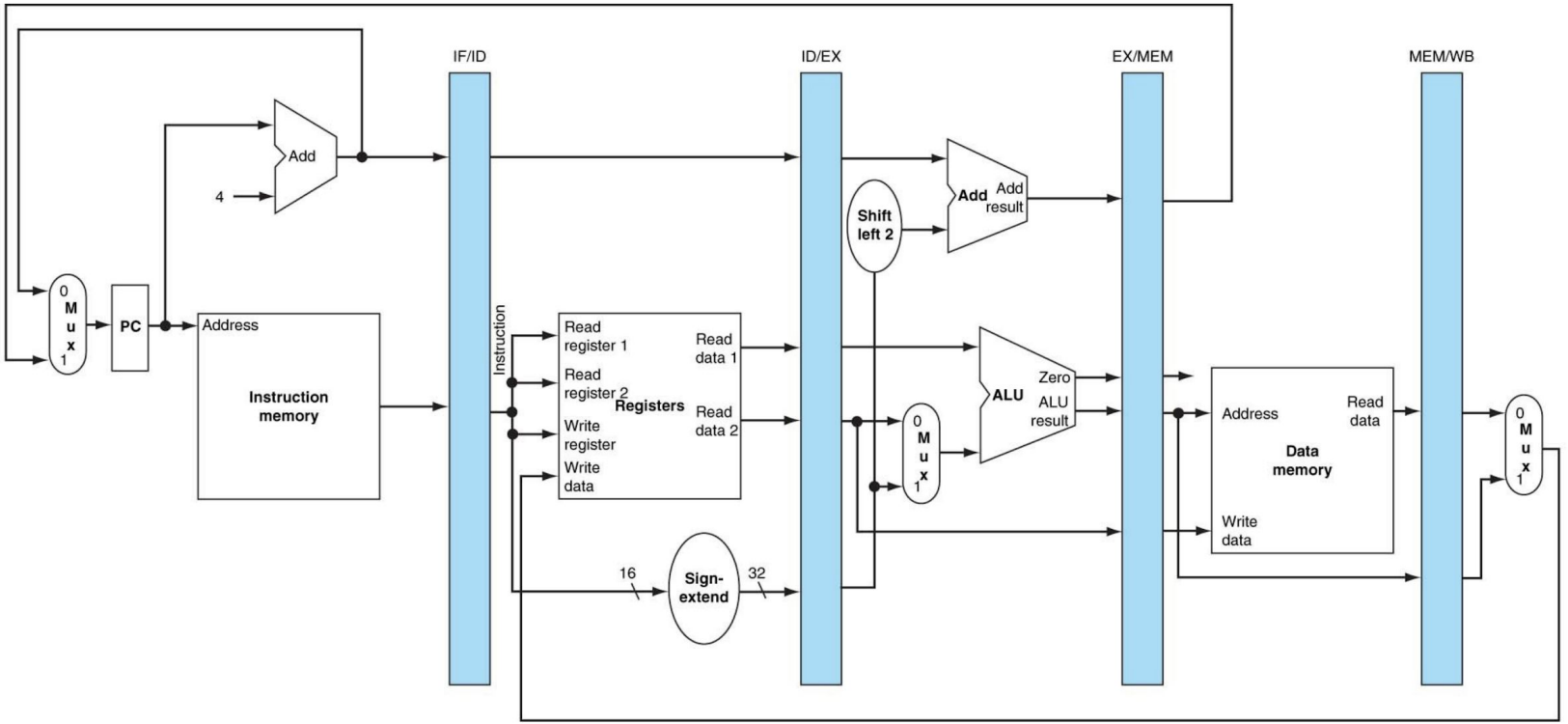
Put **lrwspc** before replaced instructions, and get below benefits:

- No limitation to instructions' type
  - PC relative ALU instruction
  - Branch instruction
  - ecall (Uprobe single_step)
- Simpler jump back instruction
- lrwspc could be improved by prediction (similar to BTB in dynamic branch prediction)

lrw2spc, imm -> spc = Mem[pc +/- imm]

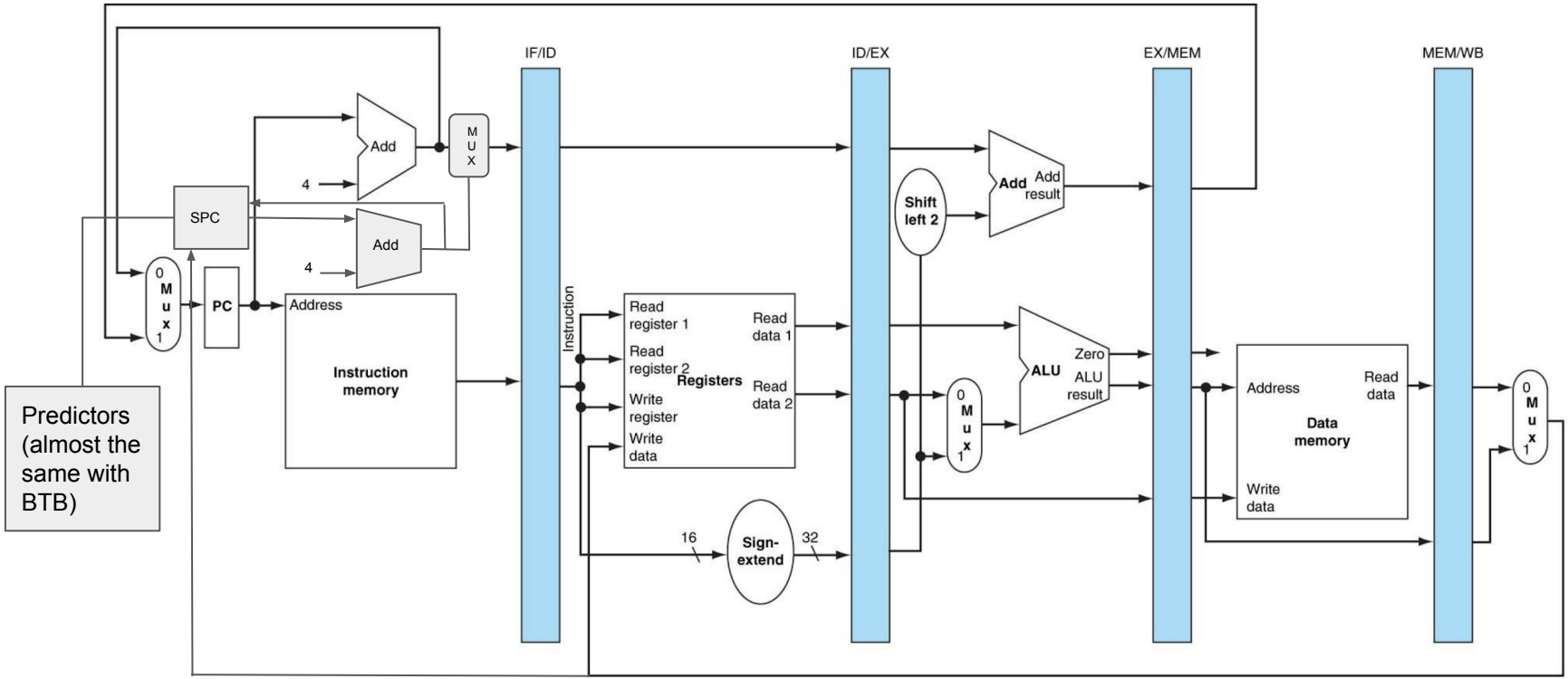
<Detour buffer>:
optprobe_template_restore_begin:
    restore regs
optprobe_template_restore_orig_insn:
    lrwspc, &constant_val
    <replaced instructions>
    jump back to origin exec_path
...
Constant_val:
    64bit value

Easy for hardware implementation



https://web.ece.ucsb.edu/~strukov/ece154aFall2013/viewgraphs/pipelinedMIPS.pdf

Easy for hardware implementation



Predictors (almost the same with BTB)

Benefit:

- Prevent a mounts of modifying pt_regs simulation codes which modifying pt_regs
- Better Performance

Q: From Linux kprobe & ftrace view, Is it valuable to be implemented? (If no, please shout it out :)

Thank you

RISC-V Linux Tracing (K/Uprobe)

26 Aug 2020, 08:00

30m

Microconference3/Virtual-Room (LPC 2020)

Speaker

Mr Guo Ren

T-HEAD

RISC-V Linux Tracing (K/Uprobe)

Guo Ren
<guoren@kernel.org>
<guoren@linux.alibaba.com>

Self Introduction

- Focus on T-HEAD Xuantie CPUs' linux porting
  - RV64GCV:
    - C910: 12-stage pipeline, 3-issue, 8-execution, SMP with multi-clusters
  - C-SKY 32bit:
    - C807: 8-stage pipeline, 2-issue, low power
    - C810: 10-stage pipeline, 2-issue, 5-execution
    - C860: 12-stage pipeline, 3-issue, 8-execution, SMP

More details ref to t-head

- Linux/arch/csky subsystem maintainer

My recent work

- Author of csky's ftrace, k/uprobe
  - 8f6bb79 csky: Add uprobes support
  - 33e53ae csky: Add kprobes supported
  - 28bb030 csky/ftrace: Add dynamic function tracer with call-graph
  - 89a3927 csky: Implement ftrace with regs
  - d7950be csky: ftrace call graph supported
  - 230c77a csky: basic ftrace supported
- The one of contributors for riscv k/uprobe
  - [PATCH v3 4/7] riscv: Add kprobes supported
  - [PATCH v3 5/7] riscv: Add uprobes supported
  - [PATCH v3 6/7] riscv: Add KPROBES_ON_FTRACE supported
  - [PATCH v3 7/7] riscv: Add support for function error injection

Dependent work

- [RFC/RFT 2/2] RISC-V: kprobes/kretprobe support (by Patrick Stählin 2018-11-13)
- [PFC/RFT 1/2] RISC-V: Implement ptrace regs and stack API
- riscv/ftrace: Add basic support (by Alan Kao 2017-12-18)
- riscv/ftrace: Add dynamic function tracer support (by Alan Kao 2018-01-13)
- riscv/ftrace: Add DYNAMIC_FTRACE_WITH_REGS support (by Alan Kao 2018-01-13)

- riscv: introduce interfaces to patch kernel code (by Zong Li 2020-04-08)

Feature support statistics in linux-5.8

- Archs with K&UPROBES:
  (powerpc, arm32/64, s390, sparc, x86, parisc, csky, mips)

- Archs with LIVEPATCH:
  (powerpc, s390, x86)

- Archs with OPTPROBES:
  (powerpc, arm32, x86)

GOAL: Let RISC-V & C-SKY support the features above this year

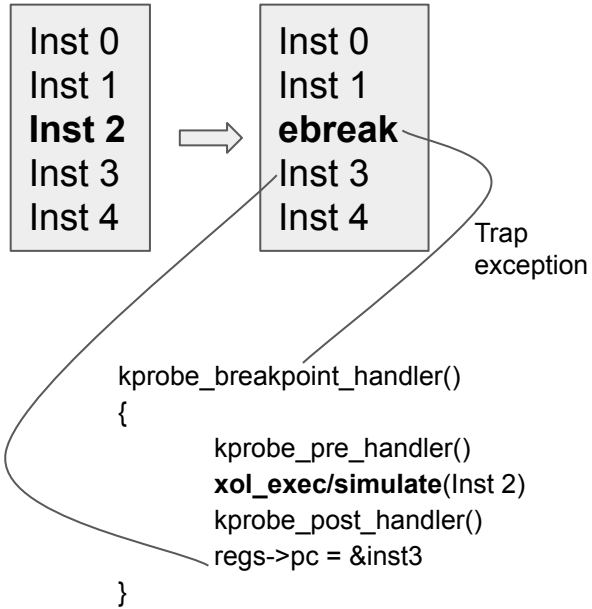
Agenda

- Ongoing work
  - Kprobe (done)
    - Single_step_exe (done)
    - Simulate_exe (done)
    - trampoline_direct_exe in Optprobes
  - Kretprobe (done)
  - Uprobe (done)
  - Kprobe on ftrace (done)
  - Livepatch
  - Optprobes
    - x86, arm, powerpc
    - Puzzles of riscv & csky, solution discussion
- Demo

Kprobe

How probe ‘inst 2’ to a kprobe point work?

- Replace ‘Inst 2’ with ebreak
- When any hart met ebreak, the ebreak TRAP exception was caused
- Then kprobe_breakpoint_handler() will call pre_handler, emulate ‘Inst 2’, and post_handler
  (eg: Error Injection, tracing event, bpf, perf point)
- Return to Inst 3, continue ...

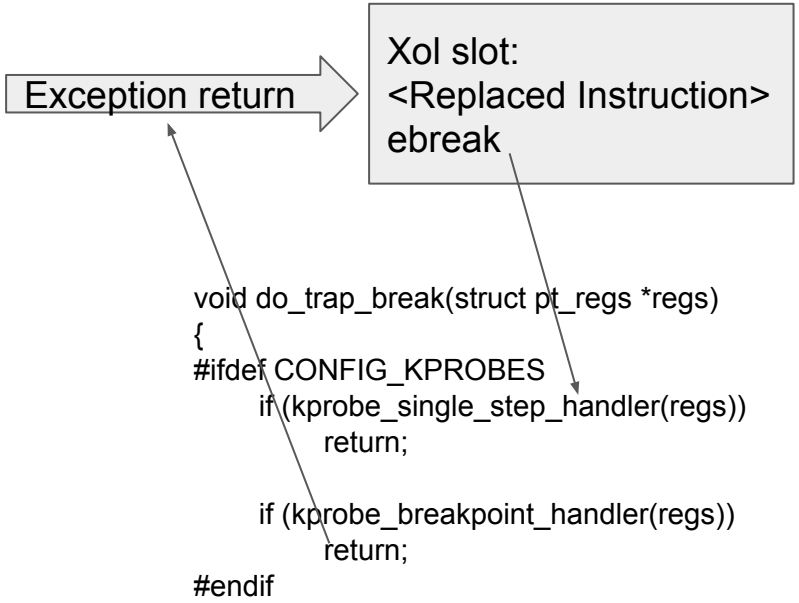


Emulate replaced instruction (xol/simulate/opt_exe)

- Use ‘ebreak’ to replace target instruction, and emulate the replaced instruction at another place
- Q: Why not put the instruction back to origin place to singlestep?
  A: SMP! Prevent other harts from missing the probe point
- Conclusion 3 methods to execute replaced instruction:
  - xol_ss_exe: Single step the replaced instruction at other place
  - Simulate_exe: modifying the contents of the pt_regs
  - xol_direct_exe: Pending replaced instruction(s) at the end of detour progress before final trampoline back

Single step execute the replaced instruction

- RISC-V privileged ISA hasn't single step exception, so the implementation is a little different from other architectures
- To simulate single step mechanism:
  - Prepare a bigger slot to hold ebreak instruction behind target instruction
  - In ebreak exception handler, call kprobe_single_step_handler(regs) (Kprobe state machine framework is well done)



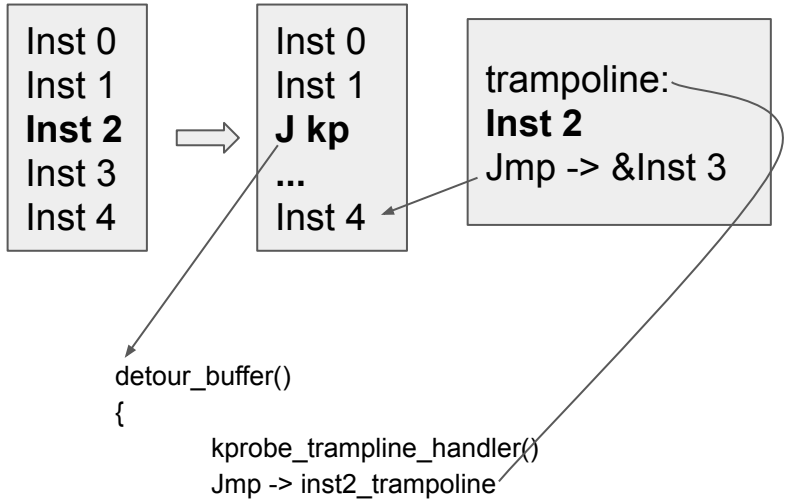
Simulate replaced instruction

- Q: Why simulate?
  - Some instructions couldn’t be single-step emulated:
    - auipc
    - branch/beqz/bnez
    - jal/j/jalr/jr
  - Some instructions must be rejected to kprobe:
    - csrrw/csrrs/csrrc/csrrwi/csrrsi/csrrci
    - fence/sfence.vma
    - ecall/ebreak
    - lr/sc sequence
- Q: How to simulate?
  A: Modify the values of pt_regs in stack

Pending replaced instruction(s) with trampoline

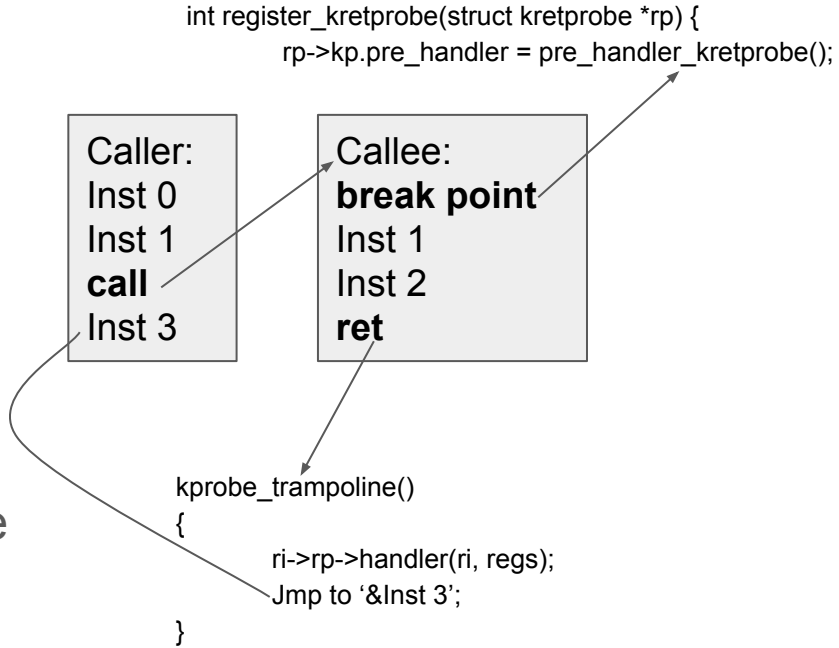
An optimized way to execute replaced instruction used by OPTPROBES

I've talked about this in another topic - [Kprobes Jump Optimized for more Archs]



Kretprobe

- Hijack the caller's return address to kretprobe_trampoline which stored in stack by pre_handler_kretprobe()
- Instead of returning to the parent caller's next instruction, it returns to kretprobe_trampoline()
- In kretprobe_trampoline(), it could handle any kinds of hook function
- Return from kretprobe_trampoline to parent caller

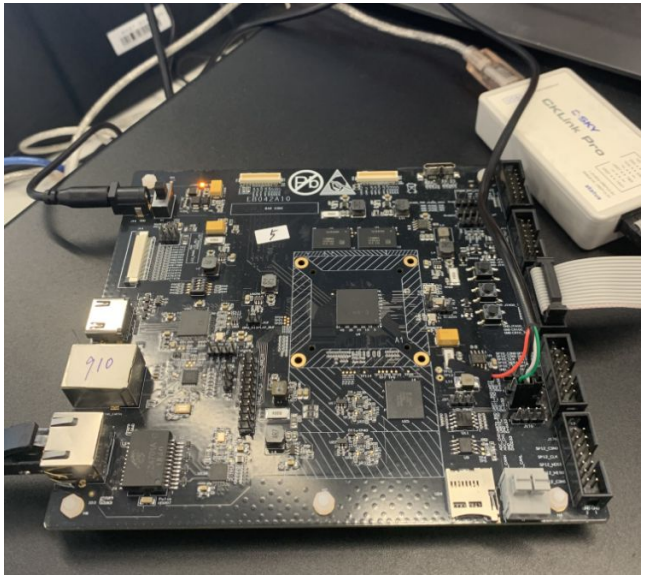


Uprobe

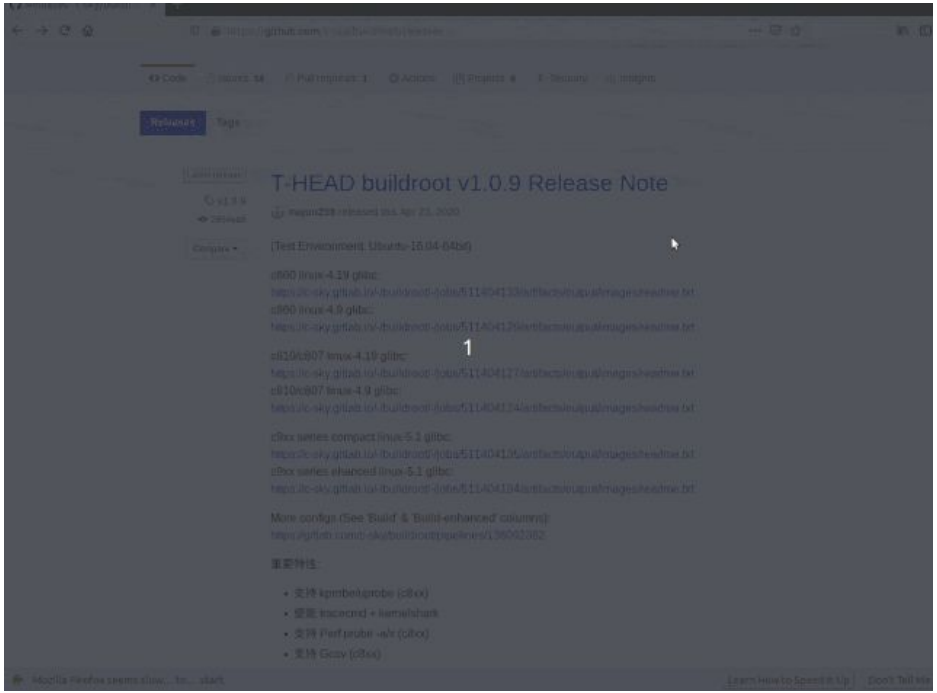
- Similar to kprobe:
  - Uprobe
  - Uretprobe
- Similar to kprobe emulate replaced instruction:
  - Singlestep replaced instruction
  - Simulate replaced instruction
- Prepare user space vma of current->mm for single step execution slot
- Not found like detour mechanism for optimizing (Any opinions here?)

Demo

https://github.com/c-sky/buildroot/releases
https://occ.t-head.cn/



Testchip: 810 + 860*4 + 910f*2 + 910v



Kprobe on ftrace

- If kprobe point is on the ftrace call site, we could utilize ftrace detour mechanism to process kprobe handler.
- Performance benefit - prevent break point,then the ftrace way is much faster.
- ftrace RISC-V by Alan on Youtube

```
# cat /sys/kernel/debug/kprobes/list
(current)
ffffffe00020af7e  k  _do_fork+0x1a    [FTRACE]

(should be)
ffffffe00020af7e  k  _do_fork+0x0     [FTRACE]
```

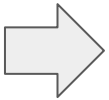
Suggested by Masami (Use -fpatchable-function-entry in ftrace) ref: mailing list. Now, I agree with that and it should be implemented immediately.

Current ftrace detour mechanism by Alan

-pg -fno-omit-frame-pointer -fno-optimize-sibling-calls -Wl,--no-relax :

000000000001065c <funca>:

```
1065c:    1141          addi    sp,sp,-16
1065e:    e406          sd      ra,8(sp)
10660:    e022          sd      s0,0(sp)
10662:    0800          addi    s0,sp,16
10664:    8786          mv      a5,ra
10666:    853e          mv      a0,a5
10668:    00000097      auipc   ra,0x0 -> nop
1066c:    ed8080e7      jalr    -296(ra) <_mcount@plt> -> nop
10670:    000127b7      lui     a5,0x12
10674:    06c7a783      lw      a5,108(a5) # 1206c <a>
10678:    2785          addiw   a5,a5,1
1067a:    0007871b      sext.w  a4,a5
1067e:    000127b7      lui     a5,0x12
10682:    06e7a623      sw      a4,108(a5) # 1206c <a>
10686:    4781          li      a5,0
10688:    853e          mv      a0,a5
1068a:    60a2          ld      ra,8(sp)
1068c:    6402          ld      s0,0(sp)
1068e:    0141          addi    sp,sp,16
10690:    8082          ret
```



When enable ftrace, replace nop with jmp ftrace_xxx_caller instructions:

000000000001065c <funca>:

```
1065c:    1141          addi    sp,sp,-16
1065e:    e406          sd      ra,8(sp)
10660:    e022          sd      s0,0(sp)
10662:    0800          addi    s0,sp,16
10664:    8786          mv      a5,ra
10666:    853e          mv      a0,a5
10668:    00010001      nop -> auipc ra, 0x
1066c:    00010001      nop -> jalr ftrace_(regs_)caller
10670:    000127b7      lui     a5,0x12
10674:    06c7a783      lw      a5,108(a5) # 1206c <a>
10678:    2785          addiw   a5,a5,1
1067a:    0007871b      sext.w  a4,a5
1067e:    000127b7      lui     a5,0x12
10682:    06e7a623      sw      a4,108(a5) # 1206c <a>
10686:    4781          li      a5,0
10688:    853e          mv      a0,a5
1068a:    60a2          ld      ra,8(sp)
1068c:    6402          ld      s0,0(sp)
1068e:    0141          addi    sp,sp,16
10690:    8082          ret
```

-fpatchable-function-entry, solution 1:

00000000000103fe <funca>:

```
103fe:    0001          nop
10400:    0001          nop
10402:    00010001      nop, nop
10406:    00010001      nop, nop
1040a:    0001          nop
1040c:    0001          nop
1040e:    1141          addi    sp,sp,-16
10410:    e422          sd      s0,8(sp)
10412:    0800          addi    s0,sp,16
10414:    8301a783      lw      a5,-2000(gp) # 12030 <a>
10418:    2785          addiw   a5,a5,1
1041a:    0007871b      sext.w  a4,a5
1041e:    82e1a823      sw      a4,-2000(gp) # 12030 <a>
10422:    4781          li      a5,0
10424:    853e          mv      a0,a5
10426:    6422          ld      s0,8(sp)
10428:    0141          addi    sp,sp,16
1042a:    8082          ret
```

00000000000103fe <funca>:

```
103fe:    1141          addi sp, sp, -8
10400:    e406          sd, ra, 0(sp)
10402:    00000097      auipc ra, 0x
10406:    ed80xxxx      jalr <ftrace_xxx_caller>
1040a:    60a2          ld ra, 0(sp)
1040c:    0141          addi sp, sp, 8
1040e:    1141          addi    sp,sp,-16
10410:    e422          sd      s0,8(sp)
10412:    0800          addi    s0,sp,16
10414:    8301a783      lw      a5,-2000(gp) # 12030 <a>
10418:    2785          addiw   a5,a5,1
1041a:    0007871b      sext.w  a4,a5
1041e:    82e1a823      sw      a4,-2000(gp) # 12030 <a>
10422:    4781          li      a5,0
10424:    853e          mv      a0,a5
10426:    6422          ld      s0,8(sp)
10428:    0141          addi    sp,sp,16
1042a:    8082          ret
```

-fpatchable-function-entry, solution 2:

```
00000000000103fe <funca>:
  103fe:    0001            nop
  10400:    0001            nop
  10402:    00010001        nop, nop
  10406:    00010001        nop, nop
  1040a:    0001            nop
  1040c:    0001            nop
  1040e:    1141            addi    sp,sp,-16
  10410:    e422            sd      s0,8(sp)
  10412:    0800            addi    s0,sp,16
  10414:    8301a783        lw      a5,-2000(gp) # 12030 <a>
  10418:    2785            addiw   a5,a5,1
  1041a:    0007871b        sext.w  a4,a5
  1041e:    82e1a823        sw      a4,-2000(gp) # 12030 <a>
  10422:    4781            li      a5,0
  10424:    853e            mv      a0,a5
  10426:    6422            ld      s0,8(sp)
  10428:    0141            addi    sp,sp,16
  1042a:    8082            ret
```

```
00000000000103fe <funca>:
  103fe:    0001            nop
  104fe:    xxxx            mv, x9, 0(sp)
  10402:    00000097        auipc ra, xxxxx
  10406:    ed80xxxx        jalr <ftrace_xxx_caller>
  1040a:    0001            nop
  1040c:    0001            nop
  1040e:    1141            addi    sp,sp,-16
  10410:    e422            sd      s0,8(sp)
  10412:    0800            addi    s0,sp,16
  10414:    8301a783        lw      a5,-2000(gp) # 12030 <a>
  10418:    2785            addiw   a5,a5,1
  1041a:    0007871b        sext.w  a4,a5
  1041e:    82e1a823        sw      a4,-2000(gp) # 12030 <a>
  10422:    4781            li      a5,0
  10424:    853e            mv      a0,a5
  10426:    6422            ld      s0,8(sp)
  10428:    0141            addi    sp,sp,16
  1042a:    8082            ret
```

(similar to arm64 x9-x17)

Livepatch

- Livepatch is based on the dynamic ftrace
- Fixup DYNAMIC_FTRACE_WITH_REGS to support modifying return address
- Simply enable HAVE_LIVEPATCH in Kconfig, then it could work
- Depends on objtools(x86) & HAVE_RELIABLE_STACKTRACE to enable best consistency model approach

(ref: https://www.kernel.org/doc/html/latest/livepatch/livepatch.html)