QEMU 支持扩展指令集的情况及添加扩展方法

黄大军 2024/3/20

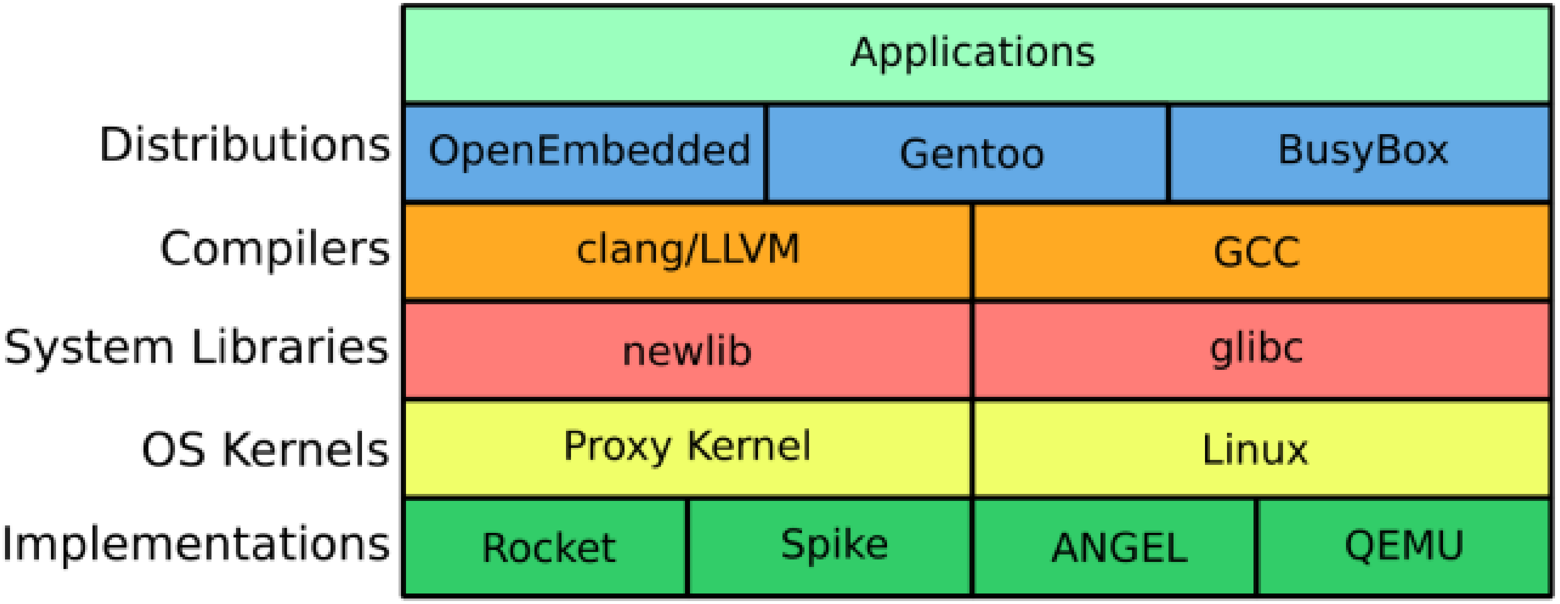
目录

- RISC-V 及 QEMU 简介
- QEMU 目前支持的 RISC-V 扩展指令集
- 为 QEMU 添加新的指令集支持

RISC-V 及 QEMU 简介

- 一个开源的基于RISC(精简指令集计算机)的指令集架构
- 模块化，低功耗，架构简单
- 基本指令集：32位(RV32I,RV32E)，64位(RV64I)，128位(RV128I)
- 可选择的扩展指令集M，A，F，D，C等
- 一个完整的64位计算机一般需要RV64GC(RV64IMAFDC的简称)

RISC-V 及 QEMU 简介



	Applications
Distributions	OpenEmbedded Gentoo BusyBox
Compilers	clang/LLVM GCC
System Libraries	newlib glibc
OS Kernels	Proxy Kernel Linux
Implementations	Rocket Spike ANGEL QEMU

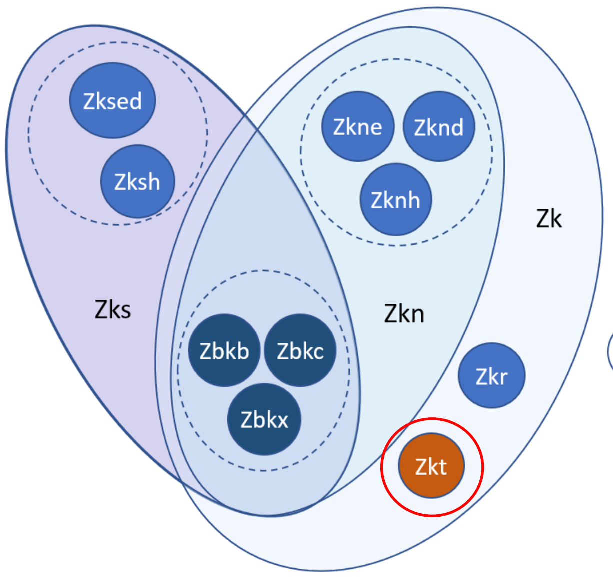
QEMU 的翻译方式为基于 Tiny Code Generator 的动态二进制翻译,即在程序运行的过程中将指令通过 TCG前端翻译成中间指令(TCG ops),再通过 TCG 后端将中间指令翻译成宿主机上可以直接运行的指令。

QEMU 支持的 RISC-V 扩展指令集

- **支持**：:MAFDCBKVHSJ, Zifencei, Zihintpause, Zicsr, Zc*, Zfinx, Zdinx, Zhinx, Zhinxmin, Zicntr, Zicbom, Zicboz, Zicbop, Zawrs, Zfa, Zfh, Ztso, Zmmul, …
- **暂不支持**：Q ✕ L ✕ T ✕ P ✕ Zihintntl ✕ Zcmop ✕ …

QEMU 支持的 RISC-V 扩展指令集

K extension



Zk* Group of instructions/CSRs defined in the cryptographic extensions chapter

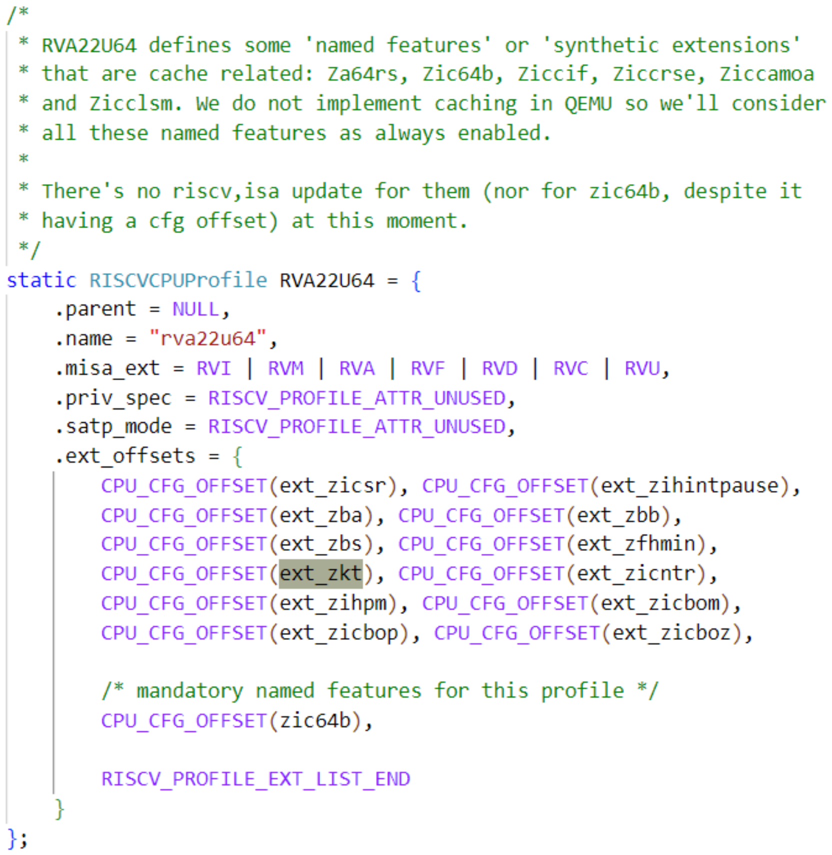
Zbk* Group of instructions defined in the bit-manipulation extensions chapter, used for cryptography

Zkt Timing behavior restricted for additional listed instructions (those not in any Zk* or Zbk* group[1])

Zk* Alias for naming a group comprising multiple smaller instruction groups commonly used together

Unnamed group of related instructions

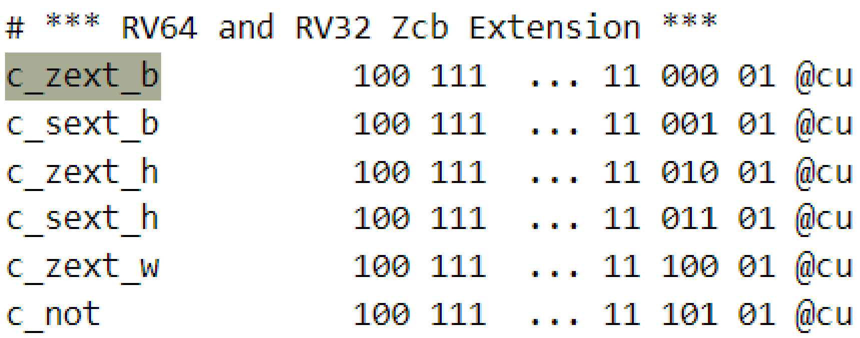
[1] Instructions instantiated using a Zk* or Zbk* group are defined as having data-independent timing; therefore, they need not be listed in Zkt



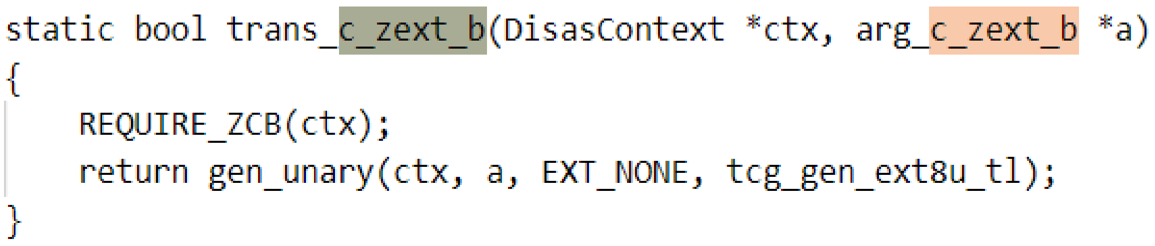
```
/*
 * RVA22U64 defines some 'named features' or 'synthetic extensions'
 * that are cache related: Za64rs, Zic64b, Ziccif, Ziccrse, Ziccamoa
 * and Zicclsm. We do not implement caching in QEMU so we'll consider
 * all these named features as always enabled.
 *
 * There's no riscv,isa update for them (nor for zic64b, despite it
 * having a cfg offset) at this moment.
 */
static RISCVCPUProfile RVA22U64 = {
    .parent = NULL,
    .name = "rva22u64",
    .misa_ext = RVI | RVM | RVA | RVF | RVD | RVC | RVU,
    .priv_spec = RISCV_PROFILE_ATTR_UNUSED,
    .satp_mode = RISCV_PROFILE_ATTR_UNUSED,
    .ext_offsets = {
        CPU_CFG_OFFSET(ext_zicsr), CPU_CFG_OFFSET(ext_zihintpause),
        CPU_CFG_OFFSET(ext_zba), CPU_CFG_OFFSET(ext_zbb),
        CPU_CFG_OFFSET(ext_zbs), CPU_CFG_OFFSET(ext_zfhmin),
        CPU_CFG_OFFSET(ext_zkt), CPU_CFG_OFFSET(ext_zicntr),
        CPU_CFG_OFFSET(ext_zihpm), CPU_CFG_OFFSET(ext_zicbom),
        CPU_CFG_OFFSET(ext_zicbop), CPU_CFG_OFFSET(ext_zicboz),

        /* mandatory named features for this profile */
        CPU_CFG_OFFSET(zic64b),

        RISCV_PROFILE_EXT_LIST_END
    }
};
```

调查方法

- 搜索源码看没有实现

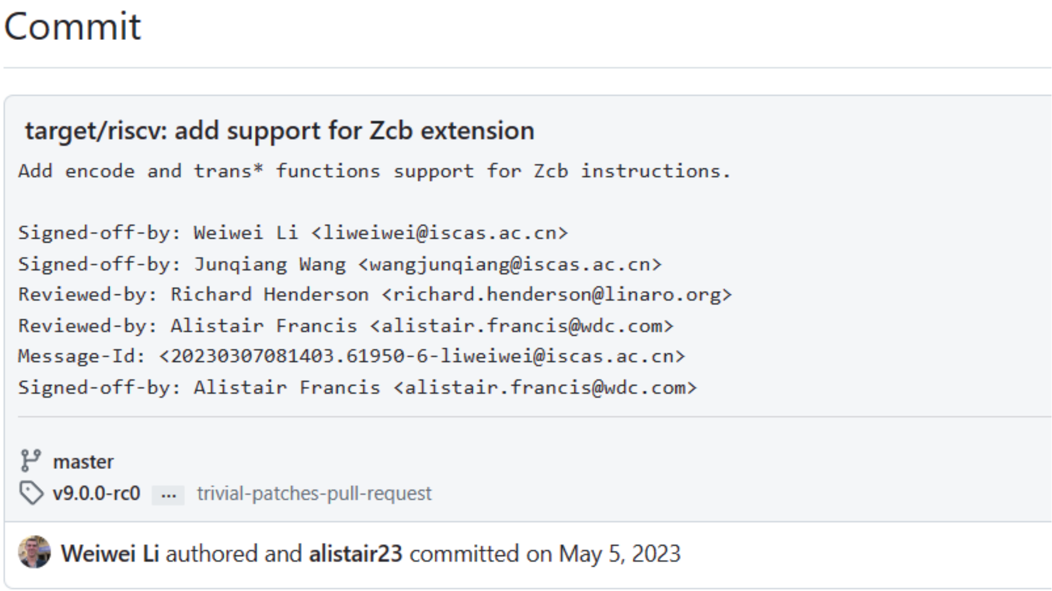


```
# *** RV64 and RV32 Zcb Extension ***
c_zext_b          100 111  ... 11 000 01 @cu
c_sext_b          100 111  ... 11 001 01 @cu
c_zext_h          100 111  ... 11 010 01 @cu
c_sext_h          100 111  ... 11 011 01 @cu
c_zext_w          100 111  ... 11 100 01 @cu
c_not             100 111  ... 11 101 01 @cu
```



```
static bool trans_c_zext_b(DisasContext *ctx, arg_c_zext_b *a)
{
    REQUIRE_ZCB(ctx);
    return gen_unary(ctx, a, EXT_NONE, tcg_gen_ext8u_tl);
}
```

- 仓库搜索有没有对应commit



Commit

target/riscv: add support for Zcb extension

```
Add encode and trans* functions support for Zcb instructions.

Signed-off-by: Weiwei Li <liweiwei@iscas.ac.cn>
Signed-off-by: Junqiang Wang <wangjunqiang@iscas.ac.cn>
Reviewed-by: Richard Henderson <richard.henderson@linaro.org>
Reviewed-by: Alistair Francis <alistair.francis@wdc.com>
Message-Id: <20230307081403.61950-6-liweiwei@iscas.ac.cn>
Signed-off-by: Alistair Francis <alistair.francis@wdc.com>
```

Weiwei Li authored and **alistair23** committed on May 5, 2023

为 QEMU 添加新的指令集支持

这里,我仿照 Zbs 指令集的 bclr 指令,为 QEMU 添加一个『只有一条 “bclt”指令的“Zbt”指令集』,“bclt”指令的功能是在 bclr 指令功能的基础上,再给结果加一。

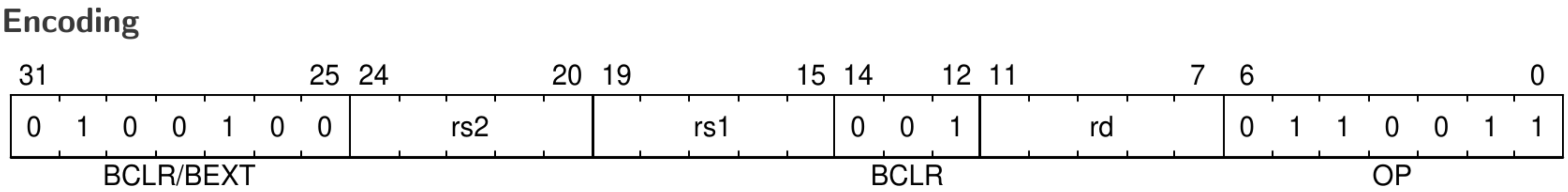
bclr rd, rs1, rs2:

```
let index = X(rs2) & (XLEN - 1);
X(rd) = X(rs1) & ~(1 << index)
```

bclr rd, 0xFFFFFFFF, 4   =>   0xFFFFFFEF

bclt rd, 0xFFFFFFFF, 4   =>   0xFFFFFFEF + 1 = 0xFFFFFFF0

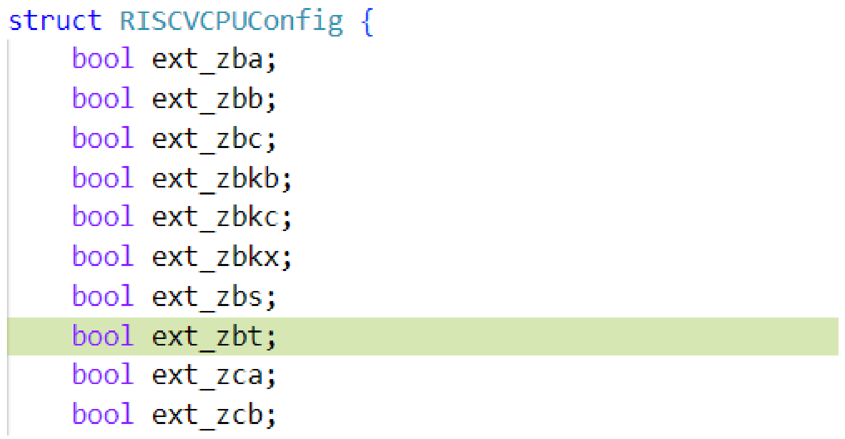
Encoding



31　　25	24　　20	19　　15	14　　12	11　　7	6　　0
0 1 0 0 1 0 0	rs2	rs1	0 0 1	rd	0 1 1 0 0 1 1
BCLR/BEXT			BCLR		OP
			0 1 1		
			“BCLT”		

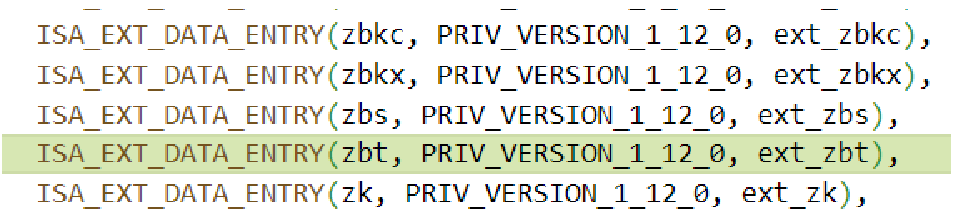
添加函数入口

- RISCVCPUConfig 结构体



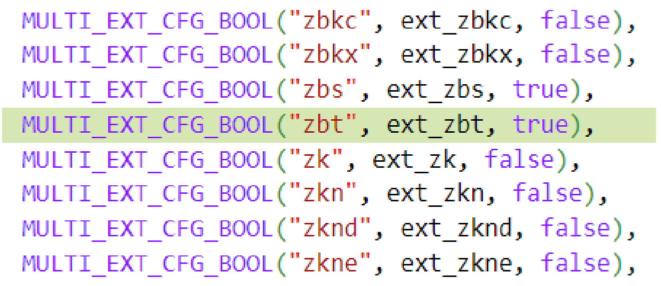
```
struct RISCVCPUConfig {
    bool ext_zba;
    bool ext_zbb;
    bool ext_zbc;
    bool ext_zbkb;
    bool ext_zbkc;
    bool ext_zbkx;
    bool ext_zbs;
    bool ext_zbt;
    bool ext_zca;
    bool ext_zcb;
```

- isa_edata_arr 结构体



```
    ISA_EXT_DATA_ENTRY(zbkc, PRIV_VERSION_1_12_0, ext_zbkc),
    ISA_EXT_DATA_ENTRY(zbkx, PRIV_VERSION_1_12_0, ext_zbkx),
    ISA_EXT_DATA_ENTRY(zbs, PRIV_VERSION_1_12_0, ext_zbs),
    ISA_EXT_DATA_ENTRY(zbt, PRIV_VERSION_1_12_0, ext_zbt),
    ISA_EXT_DATA_ENTRY(zk, PRIV_VERSION_1_12_0, ext_zk),
```

- riscv_cpu_extensions 结构体

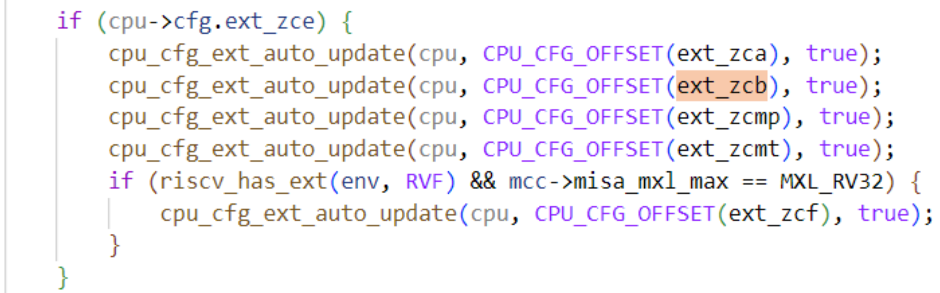


```
MULTI_EXT_CFG_BOOL("zbkc", ext_zbkc, false),
MULTI_EXT_CFG_BOOL("zbkx", ext_zbkx, false),
MULTI_EXT_CFG_BOOL("zbs", ext_zbs, true),
MULTI_EXT_CFG_BOOL("zbt", ext_zbt, true),
MULTI_EXT_CFG_BOOL("zk", ext_zk, false),
MULTI_EXT_CFG_BOOL("zkn", ext_zkn, false),
MULTI_EXT_CFG_BOOL("zknd", ext_zknd, false),
MULTI_EXT_CFG_BOOL("zkne", ext_zkne, false),
```

扩展依赖判断

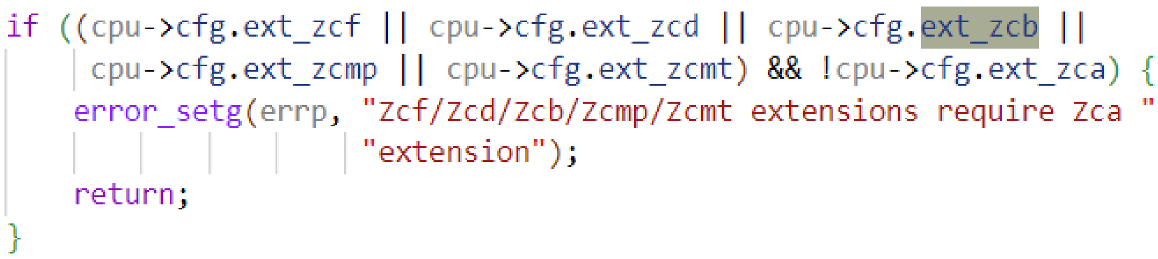
- 如果新加入的扩展对其它扩展有依赖，或拥有子扩展，在riscv_cpu_validate_set_extensions函数中判断

若包含子扩展



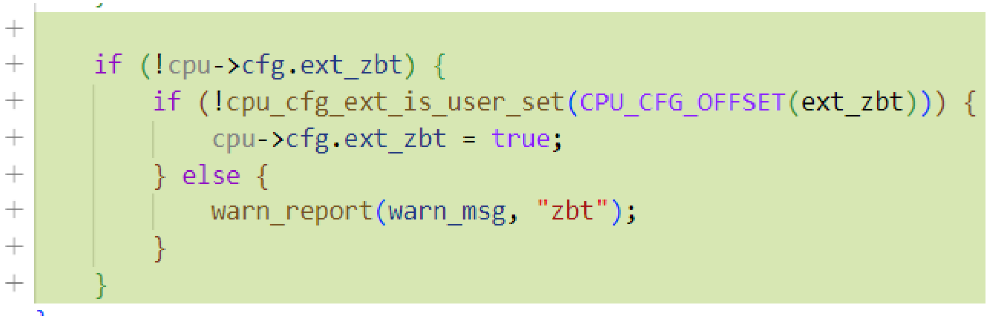
```
if (cpu->cfg.ext_zce) {
    cpu_cfg_ext_auto_update(cpu, CPU_CFG_OFFSET(ext_zca), true);
    cpu_cfg_ext_auto_update(cpu, CPU_CFG_OFFSET(ext_zcb), true);
    cpu_cfg_ext_auto_update(cpu, CPU_CFG_OFFSET(ext_zcmp), true);
    cpu_cfg_ext_auto_update(cpu, CPU_CFG_OFFSET(ext_zcmt), true);
    if (riscv_has_ext(env, RVF) && mcc->misa_mxl_max == MXL_RV32) {
        cpu_cfg_ext_auto_update(cpu, CPU_CFG_OFFSET(ext_zcf), true);
    }
}
```

若依赖于其它扩展



```
if ((cpu->cfg.ext_zcf || cpu->cfg.ext_zcd || cpu->cfg.ext_zcb ||
     cpu->cfg.ext_zcmp || cpu->cfg.ext_zcmt) && !cpu->cfg.ext_zca) {
    error_setg(errp, "Zcf/Zcd/Zcb/Zcmp/Zcmt extensions require Zca "
                     "extension");
    return;
}
```

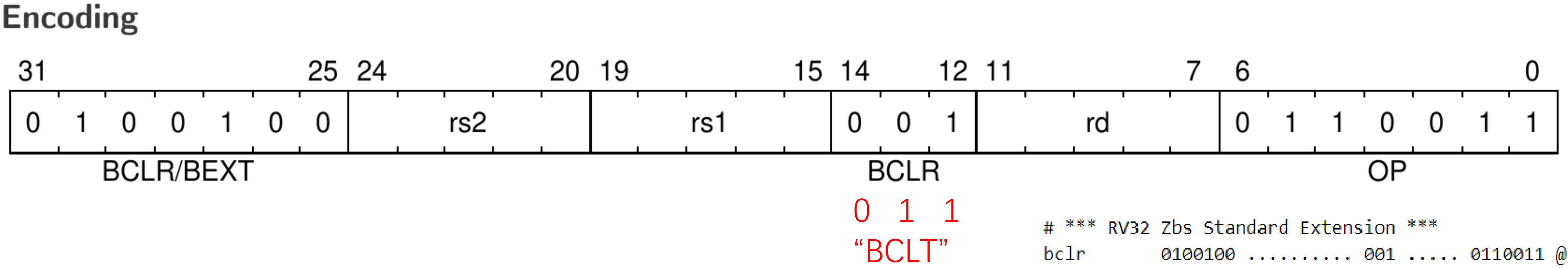
- 在riscv_cpu_validate_b 函数中添加和其有关的判断



```
+
+    if (!cpu->cfg.ext_zbt) {
+        if (!cpu_cfg_ext_is_user_set(CPU_CFG_OFFSET(ext_zbt))) {
+            cpu->cfg.ext_zbt = true;
+        } else {
+            warn_report(warn_msg, "zbt");
+        }
+    }
```

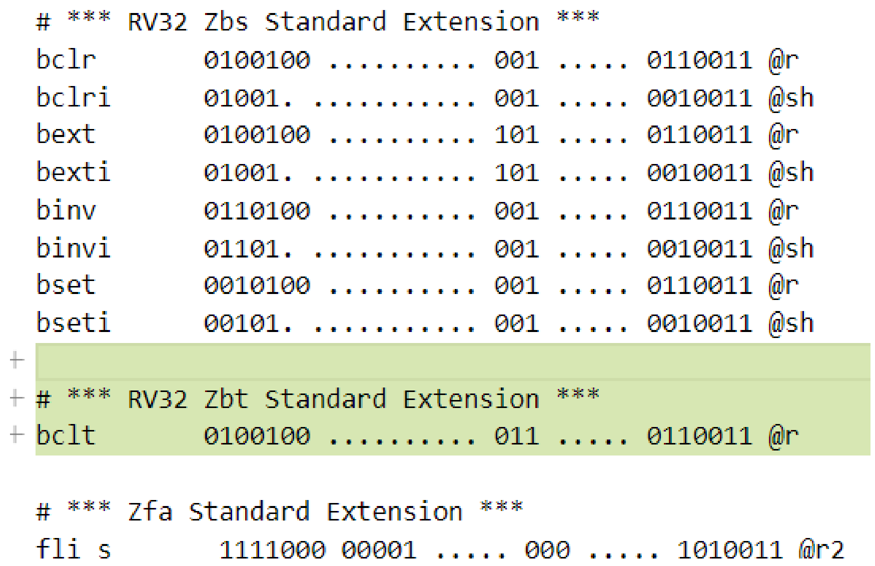
添加新扩展中指令的编码

- 在 insn32.decode 中添加新扩展中指令的编码。



Encoding

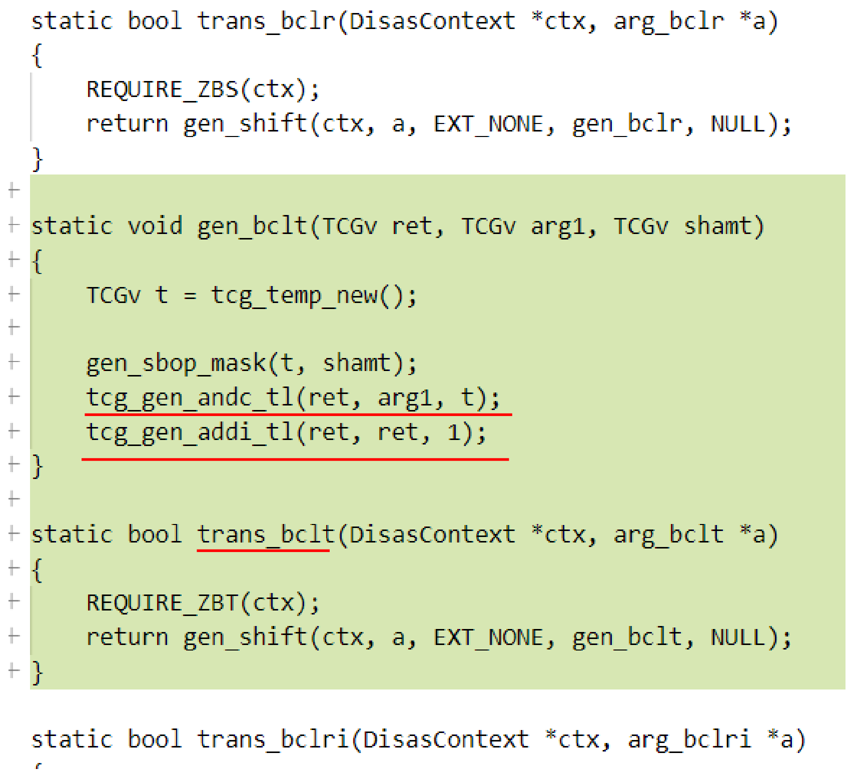
31 25	24 20	19 15	14 12	11 7	6 0
0 1 0 0 1 0 0	rs2	rs1	0 0 1	rd	0 1 1 0 0 1 1
BCLR/BEXT			BCLR		OP
			0 1 1 “BCLT”		



```
  # *** RV32 Zbs Standard Extension ***
  bclr       0100100 .......... 001 ..... 0110011 @r
  bclri      01001. ........... 001 ..... 0010011 @sh
  bext       0100100 .......... 101 ..... 0110011 @r
  bexti      01001. ........... 101 ..... 0010011 @sh
  binv       0110100 .......... 001 ..... 0110011 @r
  binvi      01101. ........... 001 ..... 0010011 @sh
  bset       0010100 .......... 001 ..... 0110011 @r
  bseti      00101. ........... 001 ..... 0010011 @sh
+
+ # *** RV32 Zbt Standard Extension ***
+ bclt       0100100 .......... 011 ..... 0110011 @r

  # *** Zfa Standard Extension ***
  fli_s        1111000 00001 ..... 000 ..... 1010011 @r2
```

添加新指令的解析函数

- 在 trans_rvb.c.inc 中添加 trans_ 开头的解析函数

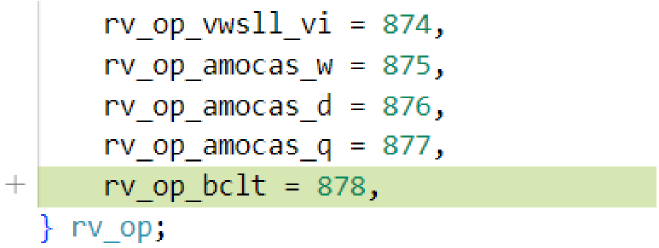


```
  static bool trans_bclr(DisasContext *ctx, arg_bclr *a)
  {
      REQUIRE_ZBS(ctx);
      return gen_shift(ctx, a, EXT_NONE, gen_bclr, NULL);
  }
+
+ static void gen_bclt(TCGv ret, TCGv arg1, TCGv shamt)
+ {
+     TCGv t = tcg_temp_new();
+
+     gen_sbop_mask(t, shamt);
+     tcg_gen_andc_tl(ret, arg1, t);
+     tcg_gen_addi_tl(ret, ret, 1);
+ }
+
+ static bool trans_bclt(DisasContext *ctx, arg_bclt *a)
+ {
+     REQUIRE_ZBT(ctx);
+     return gen_shift(ctx, a, EXT_NONE, gen_bclt, NULL);
+ }

  static bool trans_bclri(DisasContext *ctx, arg_bclri *a)
```

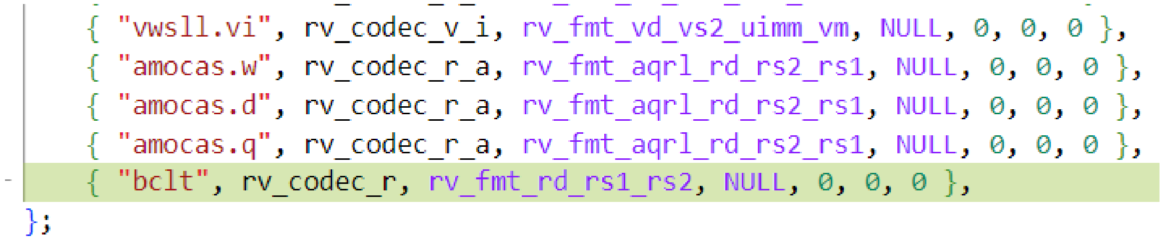
这里 trans_bclt 和 gen_bclt 调用 QEMU TCG 的函数，先进行和 bclr 指令一样的与操作，再加一。

添加反汇编支持

- 把新添加指令的入口分别加到 rv_op 结构体和 rvi_opcode_data 结构体的末尾。



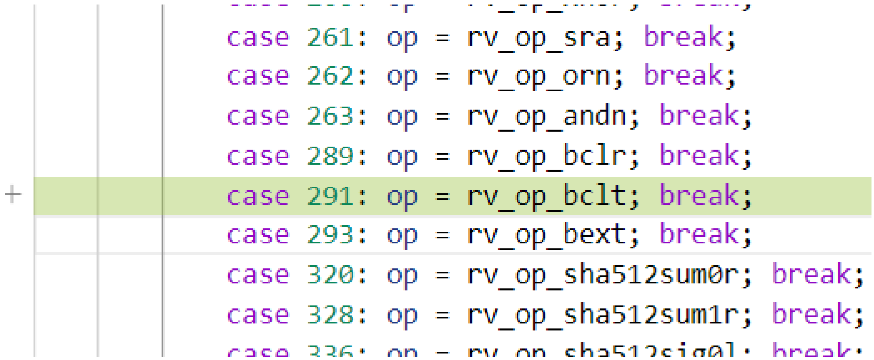
```
    rv_op_vwsll_vi = 874,
    rv_op_amocas_w = 875,
    rv_op_amocas_d = 876,
    rv_op_amocas_q = 877,
    rv_op_bclt = 878,
} rv_op;
```



```
    { "vwsll.vi", rv_codec_v_i, rv_fmt_vd_vs2_uimm_vm, NULL, 0, 0, 0 },
    { "amocas.w", rv_codec_r_a, rv_fmt_aqrl_rd_rs2_rs1, NULL, 0, 0, 0 },
    { "amocas.d", rv_codec_r_a, rv_fmt_aqrl_rd_rs2_rs1, NULL, 0, 0, 0 },
    { "amocas.q", rv_codec_r_a, rv_fmt_aqrl_rd_rs2_rs1, NULL, 0, 0, 0 },
    { "bclt", rv_codec_r, rv_fmt_rd_rs1_rs2, NULL, 0, 0, 0 },
};
```

- 在 decode_inst_opcode 函数中按照指令编码规则在合适的位置添加反汇编入口。

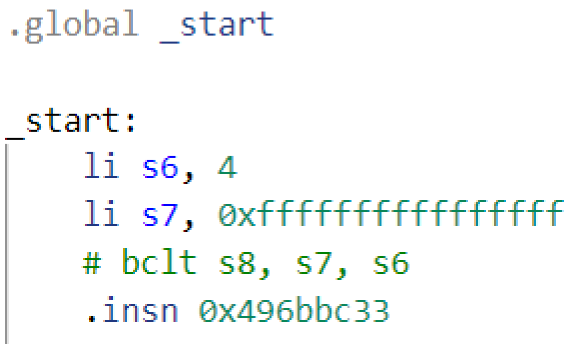
```
switch (((inst >> 22) & 0b1111111000) |
            ((inst >> 12) & 0b0000000111)) {
```



```
case 261: op = rv_op_sra; break;
case 262: op = rv_op_orn; break;
case 263: op = rv_op_andn; break;
case 289: op = rv_op_bclr; break;
case 291: op = rv_op_bclt; break;
case 293: op = rv_op_bext; break;
case 320: op = rv_op_sha512sum0r; break;
case 328: op = rv_op_sha512sum1r; break;
case 336: op = rv_op_sha512sig0l; break;
```

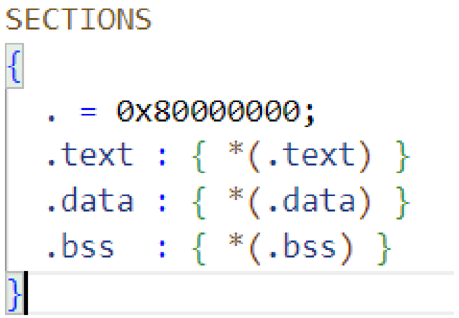
新指令功能验证

hello.s:



```
.global _start

_start:
    li s6, 4
    li s7, 0xffffffffffffffff
    # bclt s8, s7, s6
    .insn 0x496bbc33
```

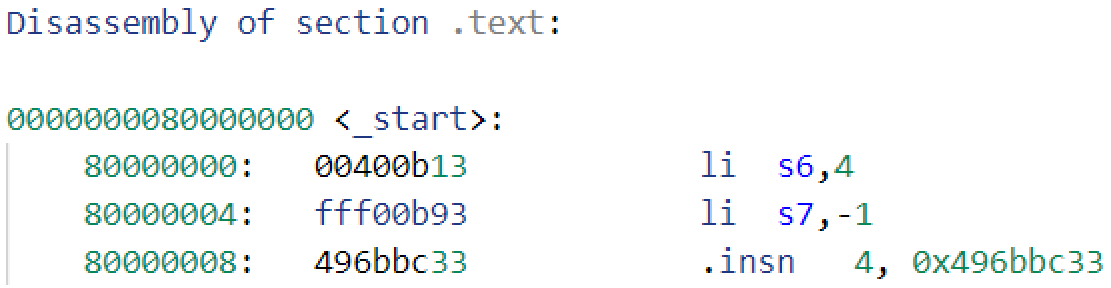
link.ld



```
SECTIONS
{
  . = 0x80000000;
  .text : { *(.text) }
  .data : { *(.data) }
  .bss  : { *(.bss) }
}
```

编译、链接、制作 QEMU 镜像命令:

```
riscv64-unknown-elf-gcc -nostartfiles -g -mabi=lp64 -march=rv64g -c hello.s -o hello.o
riscv64-unknown-elf-ld -T link.ld --no-warn-rwx-segments -o hello.elf hello.o
riscv64-unknown-elf-objcopy hello.elf -I binary hello.img
riscv64-unknown-elf-objdump -d hello.elf > hello.asm
```



```
Disassembly of section .text:

0000000080000000 <_start>:
    80000000:    00400b13        li   s6,4
    80000004:    fff00b93        li   s7,-1
    80000008:    496bbc33        .insn   4, 0x496bbc33
```

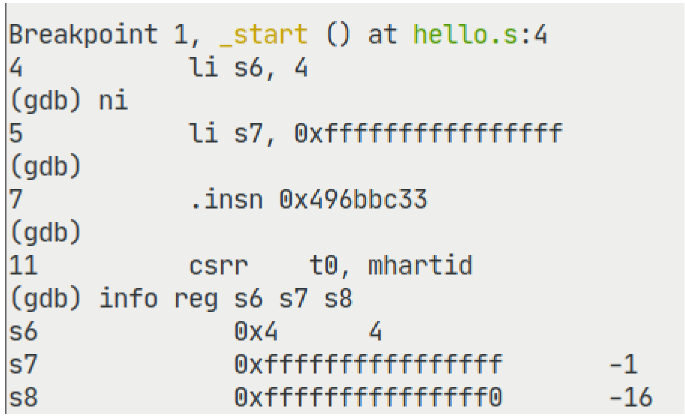
新指令功能验证

- 启动 QEMU

```
../qemu/build/qemu-system-riscv64 -cpu rv64,zbt=true -s -S -M virt -bios none -serial stdio -display none -kernel hello.img
```

- GDB 连接 QEMU 调试

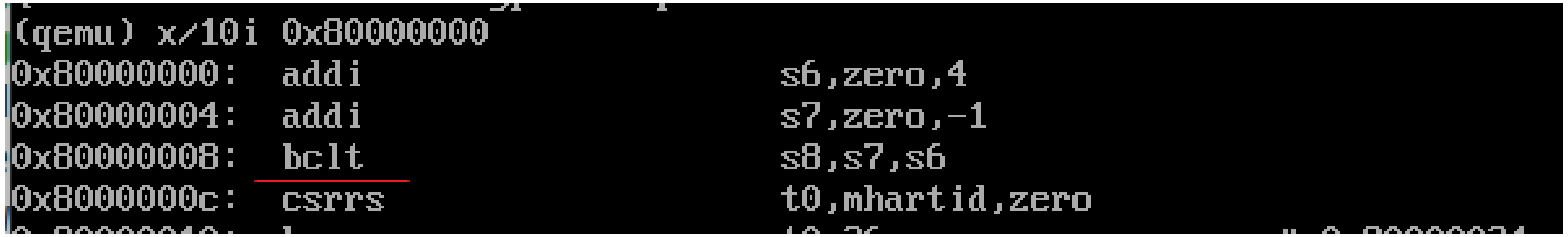
$s8 = s7 \,\&\, {\sim}(1 << s6) + 1$



```
Breakpoint 1, _start () at hello.s:4
4           li s6, 4
(gdb) ni
5           li s7, 0xffffffffffffffff
(gdb)
7           .insn 0x496bbc33
(gdb)
11          csrr    t0, mhartid
(gdb) info reg s6 s7 s8
s6             0x4      4
s7             0xffffffffffffffff       -1
s8             0xfffffffffffffff0       -16
```

新指令功能验证

QEMU monitor 反汇编功能查看内存



```
(qemu) x/10i 0x80000000
0x80000000:  addi                    s6,zero,4
0x80000004:  addi                    s7,zero,-1
0x80000008:  bclt                    s8,s7,s6
0x8000000c:  csrrs                   t0,mhartid,zero
```

Thanks