Ex 1

> Basic gdb operations cited from link

- s [NUM]: step forward [NUM] times
- si: step forward in asm mode
- c: continue
- b FUNCTION: add break points
- info breakpoints: print breakppoints
- del [NUM]: delete [NUM]th breakpoint
- layout asm/split: provide view of codes and assembly code
- p [VARIABLE]: print the information of [VARIABLE]
- x/g $[REGISTER]: look up the values in [REGISTER]

> use `make CPUS=1 qemu-gdb` and `gdb-multiarch`. Use `where` in gdb console

```
(gdb) where
#0  0x0000000000001000 in ?? ()
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
```

> riscv64-linux-gnu-readelf -h kernel/kernel

```
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           RISC-V
  Version:                           0x1
  Entry point address:               0x80000000
  Start of program headers:          64 (bytes into file)
  Start of section headers:          231840 (bytes into file)
  Flags:                             0x5, RVC, double-float ABI
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         2
  Size of section headers:           64 (bytes)
  Number of section headers:         19
  Section header string table index: 18
```

> iscv64-linux-gnu-readelf -S kernel/kernel

```
There are 19 section headers, starting at offset 0x389a0:

Section Headers:
  [Nr] Name              Type            Address          Offset
       Size              EntSize         Flags  Link  Info  Align
  [ 0]                   NULL            0000000000000000  00000000
```

```
       0000000000000000  0000000000000000           0     0     0
  [ 1] .text             PROGBITS         0000000080000000  00001000
       0000000000008000  0000000000000000  AX       0     0     16
  [ 2] .rodata           PROGBITS         0000000080008000  00009000
       0000000000000820  0000000000000000   A       0     0     8
  [ 3] .data             PROGBITS         0000000080008820  00009820
       0000000000000044  0000000000000000  WA       0     0     8
  [ 4] .got              PROGBITS         0000000080008868  00009868
       0000000000000010  0000000000000008  WA       0     0     8
  [ 5] .got.plt          PROGBITS         0000000080008878  00009878
       0000000000000010  0000000000000008  WA       0     0     8
  [ 6] .bss              NOBITS           0000000080009000  00009888
       000000000001d240  0000000000000000  WA       0     0     4096
  [ 7] .debug_info       PROGBITS         0000000000000000  00009888
       0000000000010d77  0000000000000000           0     0     1
  [ 8] .debug_abbrev     PROGBITS         0000000000000000  0001a5ff
       0000000000003475  0000000000000000           0     0     1
  [ 9] .debug_loc        PROGBITS         0000000000000000  0001da74
       0000000000009d56  0000000000000000           0     0     1
  [10] .debug_aranges    PROGBITS         0000000000000000  000277ca
       0000000000000450  0000000000000000           0     0     1
  [11] .debug_ranges     PROGBITS         0000000000000000  00027c1a
       00000000000007f0  0000000000000000           0     0     1
  [12] .debug_line       PROGBITS         0000000000000000  0002840a
       000000000000a687  0000000000000000           0     0     1
  [13] .debug_str        PROGBITS         0000000000000000  00032a91
       0000000000000f59  0000000000000001  MS       0     0     1
  [14] .comment          PROGBITS         0000000000000000  000339ea
       0000000000000029  0000000000000001  MS       0     0     1
  [15] .debug_frame      PROGBITS         0000000000000000  00033a18
       0000000000002d98  0000000000000000           0     0     8
  [16] .symtab           SYMTAB           0000000000000000  000367b0
       0000000000001908  0000000000000018          17    65     8
  [17] .strtab           STRTAB           0000000000000000  000380b8
       0000000000000837  0000000000000000           0     0     1
  [18] .shstrtab         STRTAB           0000000000000000  000388ef
       00000000000000b1  0000000000000000           0     0     1
Key to Flags:
  W (write), A (alloc), X (execute), M (merge), S (strings), I (info),
  L (link order), O (extra OS processing required), G (group), T (TLS),
  C (compressed), x (unknown), o (OS specific), E (exclude),
  p (processor specific)
```

- entry point at 0x80000000
- search `0x80000000`, and find

```
// kernel/kernel.ld
. = 0x80000000;

// kernel/memlayout.h
#define KERNBASE 0x80000000L

// entry.S
/*
    qemu -kernel loads the kernel at 0x80000000
    and causes each CPU to jump there.
    kernel.ld causes the following code to
    be placed at 0x80000000.
*/
```

> The system begins from entry.S, so place `b _entry`

```
# entry.S:18
<!-- # jump to start() in start.c -->
```

> ELF file: load and execute

- Load Memory Address，LMA
- Virtual Memory Address，VMA
- Execute

```
riscv64-linux-gnu-objdump -h kernel/kernel
```

```
kernel/kernel:     file format elf64-littleriscv

Sections:
Idx Name          Size      VMA               LMA               File off  Algn
  0 .text         00008000  0000000080000000  0000000080000000  00001000  2**4
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .rodata       00000820  0000000080008000  0000000080008000  00009000  2**3
                  CONTENTS, ALLOC, LOAD, READONLY, DATA
  2 .data         00000044  0000000080008820  0000000080008820  00009820  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  3 .got          00000010  0000000080008868  0000000080008868  00009868  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  4 .got.plt      00000010  0000000080008878  0000000080008878  00009878  2**3
                  CONTENTS, ALLOC, LOAD, DATA
  5 .bss          0001d240  0000000080009000  0000000080009000  00009888
2**12
                  ALLOC
  6 .debug_info   00010d77  0000000000000000  0000000000000000  00009888  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  7 .debug_abbrev 00003475  0000000000000000  0000000000000000  0001a5ff  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  8 .debug_loc    00009d56  0000000000000000  0000000000000000  0001da74  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
  9 .debug_aranges 00000450  0000000000000000  0000000000000000  000277ca
2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
```

```
 10 .debug_ranges 000007f0  0000000000000000  0000000000000000  00027c1a  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 11 .debug_line   0000a687  0000000000000000  0000000000000000  0002840a  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 12 .debug_str    00000f59  0000000000000000  0000000000000000  00032a91  2**0
                  CONTENTS, READONLY, DEBUGGING, OCTETS
 13 .comment      00000029  0000000000000000  0000000000000000  000339ea  2**0
                  CONTENTS, READONLY
 14 .debug_frame  00002d98  0000000000000000  0000000000000000  00033a18  2**3
                  CONTENTS, READONLY, DEBUGGING, OCTETS
```

```
// kernel/kernel.ld
. = ALIGN(0x1000); // align the memory

// not happened if different VMA and LMA
.text KERNEL_VADDR + init_end : AT(init_end) {
    *(.text*)
}
```

> stack initialization

```
// entry.S
# sp = stack0 + (hartid * 4096)
la sp, stack0
li a0, 1024*4
csrr a1, mhartid
addi a1, a1, 1
mul a0, a0, a1
add sp, sp, a0

// start.c: entry.S needs one stack per CPU.
__attribute__ ((aligned (16))) char stack0[4096 * NCPU];
```

> information about the shell from user/sh.c