■ RISC-V を Qemu で試す

Qemu で試す環境構築のメモ.

Qemu

関連ライブラリをインストールして

```
sudo apt install pkg-config libglib2.0-dev zlib1g-dev libpixman-1-dev
sudo apt install libsdl2-dev
```

ソースツリーを clone してビルド

```
git clone --recursive https://github.com/riscv/riscv-qemu.git
cd riscv-qemu
mkdir build && cd build
../configure --target-list=riscv64-softmmu,riscv32-softmmu,riscv64-linux-user,riscv32-linux-user
make -j8
sudo make install
```

これで

```
/usr/local/bin/qemu-riscv
```

などがインストールされる

ツールチェイン

関連ライブラリをインストールして

```
sudo apt-get install autoconf automake autotools-dev \
     curl libmpc-dev libmpfr-dev libgmp-dev libusb-1.0-0-dev \
     gawk build-essential bison flex texinfo gperf libtool \
     patchutils bc zlib1g-dev device-tree-compiler \
     pkg-config libexpat-dev
```

ソースツリーを clone してビルド

```
git clone https://github.com/riscv/riscv-tools.git
cd riscv-tools
git submodule update --init --recursive
export RISCV=/usr/local/riscv-tools
sudo -E ./build.sh
```

動作確認

とりあえず,

hello.s

```
        .section .text
        .equ UART_ADDR, 0x10000000

        .global _start
```

```
_start:
        la s0, message  # set a pointer to head of message

loop:
        lb a0, 0(s0)    # load head of message into a0
  addi s0, s0, 1  # increment the pointer for message
  beqz a0, halt
  jal put
  j loop

put:
        li t0, UART_ADDR
  sb a0, 0(t0)
  ret

halt:
        j halt

        .section .rodata

message:
        .ascii "Hello, RISC-V.\n\0"
```

と,

hello.ld

```
OUTPUT_ARCH("riscv")
ENTRY(_start)

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

を用意して,

```
export PATH=/usr/local/riscv-tools/bin:$PATH
riscv64-unknown-elf-gcc -march=rv64g -mabi=lp64 -nostartfiles -Thello.ld hello.s -o hello
```

とコンパイルして,

```
qemu-system-riscv64 -M virt -kernel hello -nographic
```

で実行. 終了は C-a x.

C も試す.

test.c

```
volatile char * UART =  (volatile char*)0x10000000;
int main(){
    char *mesg = "Hello RISC-V\n";
    while(*mesg != 0){
      *UART = *mesg;
      mesg++;
    }
}
```

test0.s

```
        .section .text
        .global _start

_start:
        la sp, sp_top
  jal main

halt:
        j halt
```

test.ld

```
OUTPUT_ARCH("riscv")
ENTRY(_start)

SECTIONS
{
  . = 0x80000000;
  test0 : { test0.o(.text) }
  .text   : { *(.text)   }
  .rodata : { *(.rodata) }
  .data   : { *(.data)   }
  .bss    : { *(.bss)    }
  . = ALIGN(8);
  . = . + 0x4000;
  sp_top = .;
}
```

コンパイルして,

```
riscv64-unknown-elf-gcc -march=rv64g -mabi=lp64 -o test0.o -c test0.s
riscv64-unknown-elf-gcc -march=rv64g -mabi=lp64 -mcmodel=medany -o test.o -c test.c
riscv64-unknown-elf-ld -static -nostartupfiles -T test.ld -o test test0.o test.o
```

実行

```
qemu-system-riscv64 -M virt -kernel test -nographic
```

Qemu の memory mapped I/O のアドレス

riscv-qemu/hw/riscv/virt.c に定義されてる.

```
    [VIRT_DEBUG] =    {        0x0,      0x100 },
    [VIRT_MROM] =     {     0x1000,    0x11000 },
    [VIRT_TEST] =     {   0x100000,     0x1000 },
    [VIRT_CLINT] =    {  0x2000000,    0x10000 },
    [VIRT_PLIC] =     {  0xc000000,  0x4000000 },
    [VIRT_UART0] =    { 0x10000000,      0x100 },
    [VIRT_VIRTIO] =   { 0x10001000,     0x1000 },
    [VIRT_DRAM] =     { 0x80000000,        0x0 },
```