■ RISC-V

関係の環境を再構築.

Qemu

riscv-qemu から本家にマージされたってことだった

```
git clone --recursive https://git.qemu.org/git/qemu.git
./configure \
--target-list=riscv64-softmmu,riscv32-softmmu,riscv64-linux-user,riscv32-linux-user \
--prefix=$HOME/tools/riscv
```

gcc など

crosstool-ng でインストール.
https://crosstool-ng.github.io/ から 1.24.0 をダウンロードしてビルド.

```
wget http://crosstool-ng.org/download/crosstool-ng/crosstool-ng-1.24.0.tar.bz2
tar xvf crosstool-ng-1.24.0.tar.bz2
cd crosstool-ng-1.24.0
./configure --prefix=/home/miyo/tools/crosstool-ng
export PATH=$HOME/tools/crosstool-ng/bin:$PATH
rehash
ct-ng update-samples
ct-ng list-samples | grep riscv
ct-ng riscv64-unknown-elf
vi .config # インストール先ディレクトリの CT_PREFIX_DIR を変更
           # $HOME/tools/ct-ng/ にインストールすることにした
ct-ng build
```

riscv-tools

```
git clone https://github.com/riscv/riscv-tools.git
cd riscv-tools
git submodule update --init --recursive
export PATH=$HOME/tools/ct-ng/riscv64-unknown-elf/bin:$PATH
export RISCV=$HOME/tools/riscv
./build.sh
```

動作確認

hello.s

```
  .section .text
  .equ UART_ADDR, 0x10000000 #UART
  .global _start

_start:
  la s0,message

loop:
  lb   a0,0(s0)
  addi s0,s0,1
  beqz a0,halt
  jal  putchar
  j    loop

putchar:
  li   t0,UART_ADDR
  sb   a0, 0(t0)
```

```
  ret

halt:
  j    halt

  .section .rodata

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

linker.ld

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

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

ビルドして実行

```
riscv64-unknown-elf-gcc -march=rv64i -mabi=lp64 -nostartfiles -Tlinker.ld hello.s -o hello
qemu-system-riscv64 -M virt -kernel hello -nographic
```

実行結果

```
miyo@tama:% qemu-system-riscv64 -M virt -kernel hello -nographic
qemu-system-riscv64: warning: No -bios option specified. Not loading a firmware.
qemu-system-riscv64: warning: This default will change in a future QEMU release. Please use the
-bios option to avoid breakages when this happens.
qemu-system-riscv64: warning: See QEMU's deprecation documentation for details.
Hello, RISC-V # ここまで表示されたら Ctrl-a x を入力
QEMU: Terminated
```

objdump でみてみる

```
miyo@tama:% riscv64-unknown-elf-objdump -s -d hello

hello:     file format elf64-littleriscv

Contents of section .text:
 80000000 17040000 1304c402 03050400 13041400  ................
 80000010 630c0500 ef008000 6ff01fff b7020010  c.......o.......
 80000020 2380a200 67800000 6f000000           #...g...o...
Contents of section .rodata:
 8000002c 48656c6c 6f2c2052 4953432d 560a00    Hello, RISC-V..
Contents of section .riscv.attributes:
 0000 41190000 00726973 63760001 0f000000  A....riscv......
 0010 05727636 34693270 3000               .rv64i2p0.

Disassembly of section .text:

0000000080000000 <_start>:
    80000000:  00000417            auipc  s0,0x0
    80000004:  02c40413            addi  s0,s0,44 # 8000002c <message>

0000000080000008 <loop>:
    80000008:  00040503            lb  a0,0(s0)
    8000000c:  00140413            addi  s0,s0,1
    80000010:  00050c63            beqz  a0,80000028 <halt>
    80000014:  008000ef            jal  ra,8000001c <putchar>
    80000018:  ff1ff06f            j  80000008 <loop>

000000008000001c <putchar>:
```

```
    8000001c:  100002b7            lui  t0,0x10000
    80000020:  00a28023            sb  a0,0(t0) # 10000000 <UART_ADDR>
    80000024:  00008067            ret

0000000080000028 <halt>:
    80000028:  0000006f            j  80000028 <halt>
```

objcopy して hexdump

```
miyo@tama:% riscv64-unknown-elf-objcopy -O binary hello hello.bin
miyo@tama:% hexdump hello.bin
0000000 0417 0000 0413 02c4 0503 0004 0413 0014
0000010 0c63 0005 00ef 0080 f06f ff1f 02b7 1000
0000020 8023 00a2 8067 0000 006f 0000 6548 6c6c
0000030 2c6f 5220 5349 2d43 0a56 0000
000003b
```