First steps on Aarch64

Daniel Krebs

So 18 September 2016

Category: aarch64 (https://www.daniel-krebs.net/category/aarch64.html)

Up until now, I only have been involved with OS development on x86 with the HermitCore (https://github.com/RWTH-OS/HermitCore/) project and general embedded programming of STM32 microcontrollers using xpcc (https://github.com/roboterclubaachen/xpcc/). While both is fun in it's own way, I want to explore something new: ARM Aarch64. More precisely, I'm talking about ARMv8-A (http://www.arm.com/products/processors/armv8-architecture.php) which is the most recent architecture for cell phone grade SoCs. However, ARM is also trying to push into the server market with these chips, so maybe, at the end of this journey, there will a new supported architecture for HermitCore? :)

Toolchain

We'll be using the `aarch64-linux-gnu-` toolchain for the following experiments. Since I'm using Arch Linux I can build/install from the AUR:

```
$ yaourt -S aarch64-linux-gnu-gcc aarch64-linux-gnu-gdb
$ yaourt -S qemu-arch-extra
```

This gives us a compiler, debugger and system emulator.

If you're using another OS, you might find these links helpful:

- Ubuntu PPA (https://launchpad.net/~ubuntu-toolchain-r/+archive/ubuntu/aarch64)
- Binaries by Linaro (http://releases.linaro.org/components/toolchain/binaries/latest-5/)

From asm to C

Enough of the introduction, let's get our hands dirty. There's not so much work to do get out of assembler land on Aarch64. It's enough to initialize the stack pointer register with a sane value and we're good to go calling our first C function.

```
.extern stack_top
.globl _start
_start:
    ldr x30, =stack_top
    mov sp, x30
    bl main
hang:
    b hang
```

To determine where `stack_top` is located, we need to take a look at the linkerscript.

Linkerscript

We're aiming to run our program on Qemu so we have to know where RAM and peripherals are mapped. Since we'll use the `virt` machine, let's have a look at it's implementation: hw/arm/virt.c (http://git.qemu.org/?p=qemu.git;a=blob;f=hw/arm/virt.c;h=a193b5a95b3f5ea170be1768a1a81ee305df93df;hb=HEAD#l151).

```
static const MemMapEntry a15memmap[] = {
    /* ... */
    [VIRT_UART] =               { 0x09000000, 0x00001000 },
    /* ... */
    [VIRT_MEM] =                { 0x40000000, RAMLIMIT_BYTES },
    /* ... */
};
```

The RAM will be mapped at `0x40000000`, so this is our simple linkerscript:

```
ENTRY(_start)
SECTIONS
{
 . = 0x40000000;
 .startup . : { startup.o(.text) }
 .text : { *(.text) }
 .data : { *(.data) }
 .bss : { *(.bss COMMON) }
 . = ALIGN(8);
 . = . + 0x1000; /* 4kB of stack memory */
 stack_top = .;
}
```

Saying hello in C

It's time to say hello from our favorite macro assembler: C. In order to have some visual feedback let's do the obvious. Printing "Hello World" to the screen. That is, not to a real screen, but a serial terminal. Qemu implements the PrimeCell UART (PL011) (http://infocenter.arm.com/help/topic/com.arm.doc.ddi0183f/DDI0183.pdf) so sending something is as easy as writing some characters into the memory mapped data register (`UARTDR`).

```
// Data register of first UART
volatile unsigned int * const UART0_DR = (unsigned int *) 0x09000000;

void print(const char *s) {
    while(*s != '\0') {
        *UART0_DR = (unsigned int)(*s++);
    }
}

int main() {
    print("Hello world!\n");
    return 0;
}
```

Note that address of the UART peripheral can be found the memory map from above. In our case, `UARTDR` has an offset of `0x000` .

Compile and run

The last thing that's up for today is actually running our first bare-metal program. So let's do the compiling and linking:

```
$ aarch64-linux-gnu-as -o startup.o startup.asm
$ aarch64-linux-gnu-gcc -c -o main.o main.c
$ aarch64-linux-gnu-ld -Bstatic --gc-sections -nostartfiles -nostdlib -o first-steps.elf -T link.ld main.o startup.o
```

And finally we can run it using Qemu:

```
$ qemu-system-aarch64 -machine virt -m 1024M -cpu cortex-a53 -nographic -s -kernel first-steps.elf
Hello world!
```

You can find all the code including a nice Makefile at my Github repository (https://github.com/daniel-k/aarch64-hello-world).

Acknowledgements

I didn't make up everything myself, so thanks to freedomtan (https://github.com/freedomtan/aarch64-bare-metal-qemu) and ultimately Balau (https://balau82.wordpress.com/2010/02/28/hello-world-for-bare-metal-arm-using-qemu/).

Comments

Navigation

random notes (https://www.daniel-krebs.net)

atom (https://www.daniel-krebs.net/feeds/all.atom.xml)

Author

Github (https://github.com/daniel-k)

Categories

aarch64 (1) (https://www.daniel-krebs.net/category/aarch64.html)

misc (1) (https://www.daniel-krebs.net/category/misc.html)