Sparc Barebones

From OSDev Wiki

Sparc comes with a few oddities of its own, including a non-trivial bootsector implementation. The following code will provide you a running start on SPARC, but it is one big hack, it has not been written with extension in mind, and neither has it been tested on real hardware.

Tools

You will need:

- sparc-elf-as and sparc-elf-ld, just make a cross-toolchain with the sparc-elf as target.
- dd
- mkisofs
- qemu-system-sparc (get a recent version!)

GCC might come in handy later, but it is not used in this example

Bootsector

OpenFirmware's boot sequence will try to load an application from either a generic partition or a platform-specific partition. That table is located in the first 512 bytes of the storage medium. Within that partition, it will try to look for a binary located from byte 512 to 8191. QEMU will allow boot from a.out, ELF or a linux image, whereas Sun's implementation on actual hardware is said to only support a.out, so we will have to stick to that. The binary is parsed and loaded at some location in memory, so it also needs to be a position-independent implementation. After loading the file, it will jump to the start of the text section, and pass a pointer to the openfirmware client structure as the first argument (o0 register).

For demonstration purposes, we grab our load address, calculate a pointer offset and then issue a print call to the firmware.

```
.section .text
.align 16
.global _start

_start:
        call 1f                         ! put PC on the stack.
          mov %o7, %l0                  ! use the delay slot to gra
1:      mov 0x60, %l3                   ! 0x60 is the offset to the
        add %l3, %o0, %l3               ! actual address holding th
        ld [%l3], %l5                   ! address of function

        mov dataptr - _start, %l4       ! offset to text to print
        add %l4, %l0, %l4               ! absolute address of text
```

```
          mov %l4, %o0                  ! first argument is pointer
          mov 0x0b, %o1                 ! second argument is number
          call %l5                      ! call function
            nop                         ! waste the delay slot for

  2:
          call 2b                       ! loop forever
            nop


  dataptr:
          .byte 'H', 'e', 'l', 'l', 'o', ' ', 'W', 'o', 'r', 'l',
```

There's two problems now: binutils wants to make a demand-loaded binary by default, which is something qemu's default bios really does not like. Which means we have to work around that by setting the header type manually. Since a.out is such a dumb format, we only need 8 lines of code to have ld figure out the rest. We first tell it to kick out all headers, add the a.out signature we need, then the size of .text (no location needed, it comes directly after the header), followed by the size of the headers we are not using:

```
OUTPUT_FORMAT(binary)
ENTRY(_start)

SECTIONS
{
    . = 0x3980;
    .main :
    {
        LONG(0x01030107);
        LONG(_boot_end - _boot_start);
        LONG(0);
        LONG(0);
        LONG(0);
        LONG(0);
        LONG(0);
        LONG(0);
        _boot_start = .;
        *(.text*)
        *(.rodata*)
        *(.data*)
        *(.bss*)
        _boot_end = .;
    }
}
```

Building

The bootsector can then be made with the following commands:

```
sparc-elf-as boot.S -o boot.o
sparc-elf-ld boot.o -o boot.bin -T sparcboot.ld
```

SILO remarks that the firmware only really checks the partition offsets, and throws in a sector of zeroes so that it will always look at the start of the disk for the bootblock. This is a cheap way to boot off any custom medium if you don't care about any filesystem, but it is especially useful for the CD filesystem which don't put anything near that area and just filling it up with zeroes is correct as far as the standard goes:

```
dd if=/dev/zero of=bootblock.bin bs=2048 count=4
dd if=boot.bin of=bootblock.bin bs=512 seek=1 conv=notrunc
mkisofs -o helloworld.iso -G bootblock.bin /path/to/other/cd/con
```

Start qemu afterwards:

```
qemu-system-sparc -cdrom helloworld.iso
```

Which gives you the firmware prompt, after which entering

```
boot cdrom
```

will start the code.

Categories: Sparc | Bare bones tutorials
