Exploring AArch64 assembler – Chapter 1

Oct 8, 2016 • Roger Ferrer Ibáñez • AArch64

AArch64 is a new 64 bit mode that is part of the ARMv8 architecture presented in 2011 by ARM. It has been progressively been deployed in smartphones and servers. So I think it is a good moment to learn a bit more about the assembler of this architecture.

Hardware availability

Single board computers with ARMv6/ARMv7 architecture are easily available nowadays. One of the most popular choices is the Raspberry Pi.

In contrast, single-board computers that support the 64-bit mode of ARMv8 are less common but they are slowly becoming more popular these days. For instance the Pine64, the ODROID-C2, the Dragonboard 410c, etc. Any of them will do and in general they differ on the specific System on Chip being used.

> **Note**: the Raspberry Pi 3 has a CPU (Cortex-A53) that implements the 64-bit mode of the ARMv8 architecture and technically could run a 64-bit system. But the software system provided by the Raspberry Foundation (Raspbian) is only for 32-bit and there are no official plans for a 64-bit system.

> **Update**: SUSE has a 64-bit version of its OpenSUSE distribution that can run in the Raspberry Pi 3. Arch also has a 64-bit distribution that can be installed in the RPi3.

Software alternative

Does this mean that without hardware it is not possible to play with AArch64? No! We can still do many things using a cross-toolchain and QEMU in user mode.

Example for Ubuntu 16.04

Just install QEMU and a cross-toolchain for AArch64.

```
$ sudo apt-get install qemu-user gcc-aarch64-linux-gnu
```

Now test you can run a "Hello world" written in C. Create a `hello.c` file with the following contents.

```
#include <stdio.h>

int main(int argc, char *argv[])
{
  printf("Hello AArch64!\n");
  return 0;
}
```

Now compile it with the cross-compiler for AArch64 that we have installed earlier (the **`-static`** flag is important).

```
$ aarch64-linux-gnu-gcc -static -o hello hello.c
```

Check it is a AArch64 binary.

```
$ file hello
hello: ELF 64-bit LSB executable, ARM aarch64, version 1 (SYSV), statically linked, for
```

Trying to run it should fail with some confusing error.

```
$ ./hello
-bash: ./hello: No such file or directory
```

But we can run it using the QEMU for AArch64 that we installed earlier.

```
$ qemu-aarch64 ./hello
Hello AArch64!
```

Yay!

> **Note**: If you use this option, remember always to run your programs using `qemu-aarch64`.

Our first AArch64 assembler program

Let's write a very simple program that just returns an error code of two.

```
1 // first.s
2 .text
3
4 .globl main
5
6 main:
```

```
7       mov w0, #2
8       ret
```

Let's assemble it.

```
$ aarch64-linux-gnu-as -c first.s
```

And now link it, for convenience we will use `gcc`.

```
$ aarch64-linux-gnu-gcc -static -o first first.o
```

Run it and check the return.

```
$ ./first             # or use qemu-aarch64 ./first
$ echo $?
2
```

Yay!

Let's go through each line of the code above.

```
1 // first.s
2 .text
```

Line 1 is just a comment with the name of the file used in this example. Any text in a line that follows a `//` is a comment and it is ignored. Line 2 is an assembler directive that means "now come instructions of the program". This is because we can also express data in an assembler file (data goes after a `.data` directive).

```
4 .globl main
```

This is another assembler directive that means `main` is going to be a *global* symbol. This means that when constructing the final program, this file will have the global `main` symbol that is needed by the C library to start a program.

```
6 main:
7       mov w0, #2 // w0 ← 2
8       ret        // return
```

This is the entry point of our program. Line 6 itself is just a label for the symbol `main` (that we mentioned above it was a global symbol). Lines 7 and 8 are two instructions. The first one just sets the register `w0` to be 2 (we will see what registers are in the next chapter). The second one returns from the `main`, effectively finishing our program.

When finishing a program, the contents of the register `w0` are used to determine the error code of the program. This is the reason why `echo $?` above prints `2`.

Reference documentation

Documentation for the AArch64 instruction set can be found in the ARM® Architecture Reference Manual ARMv8, for ARMv8-A architecture profile

That's all for today.

« A tiny GCC front end – Part 11

How (not) to write a C++ front end – Part 1 »

Powered by Jekyll. Theme based on whiteglass
Subscribe via RSS