ECGR 4101/5101 - Lab 1

Objective: Programming bare-metal ARM with QEMU.

QEMU is a generic and open source machine emulator and virtualizer. When used as a machine emulator, QEMU can run OSes and programs made for one machine (e.g. an ARM board) on a different machine (e.g. your own PC). By using dynamic translation, it achieves very good performance.

Outcomes: After this lab, you will be able to

- Install QEMU emulator
- Use gdb
- Understand simple GNU linker scripts
- Gain familiarity with the ARM instruction set

Install QEMU dependencies

```
sudo apt-get update
sudo apt-get install build-essential
sudo apt-get install pkg-config
sudo apt-get install git libglib2.0-dev libfdt-dev libpixman-1-dev zlib1g-dev
sudo apt-get install libsdl2-dev
```

Install latest version of QEMU

```
wget https://download.qemu.org/qemu-5.1.0.tar.xz
tar xvf qemu-5.1.0.tar.xz
cd qemu-5.1.0
./configure --target-list=arm-softmmu --enable-sdl
make
```

A cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running. In our case, we are compiling to ARM instruction set on an x86 platform (depending on your PC).

Use the GNU GCC cross compiler toolchain

1. Binutils package: This consists of the ld, as, and so on.

```
sudo apt-get install binutils-arm-none-eabi
```

2. gcc package: to compile to a bare metal arm architecture

```
sudo apt-get install gcc-arm-none-eabi
```

3. gdb debugger for command line debugging.

```
sudo apt-get install gdb-multiarch
```

Hello World on Bare-metal ARM on QEMU

Make a new Lab1 directory (mkdir Lab1).
Download the following files from Canvas - Lab1
HelloWorld.c, startup.s and HelloWorld.ld
We emulate the VersatilePB platform with an ARM926EJ-S core.

HelloWorld.c is a simple program that prints the message “Hello World" without using any standard C libraries. The volatile keyword is necessary to instruct the compiler that the memory pointed by UART0DR can change or has effects independently of the program. The assembly file startup.s sets up the stack as required by C programs. The top of the stack will be defined during linking. HelloWorld.ld is the linker script that defines where the different sections (code and data) should be placed in memory, and the size of the stack.

Execute the following commands

```
arm-none-eabi-as -mcpu=arm926ej-s -g startup.s -o startup.o
arm-none-eabi-gcc -c -mcpu=arm926ej-s -g HelloWorld.c -o HelloWorld.o
arm-none-eabi-ld -T HelloWorld.ld HelloWorld.o startup.o -o HelloWorld.elf
arm-none-eabi-objcopy -O binary HelloWorld.elf HelloWorld.bin
```

The QEMU emulator is written especially to emulate Linux guest systems; for this reason its startup procedure is implemented specifically: the -kernel option loads a binary file (usually a Linux kernel) inside the system memory starting at address 0x00010000. The emulator starts the execution at address 0x00000000, where few instructions (already in place) are used to jump at the beginning of the kernel image. The interrupt table of ARM cores usually placed at address 0x00000000 is not present since the peripheral interrupts are disabled at startup as needed to boot a Linux kernel.

From the qemu-4.1.0 directory

```
./arm-softmmu/qemu-system-arm -M versatilepb -m 128M -nographic -
kernel ../Lab1/HelloWorld.bin
```

This should print “Hello World!”
To exit QEMU press ctrl-a followed by x

Examine the disassembled contents of the HelloWorld.elf

```
arm-none-eabi-objdump -d HelloWorld.elf
```

Debug the program using gdb. QEMU implements a gdb connector using a TCP connection

```
./arm-softmmu/qemu-system-arm -M versatilepb -m 128M -nographic -s -S
-kernel ../Lab1/HelloWorld.bin
```

This command freezes the system before executing any guest code, and waits for a connection on the TCP port 1234
In another window launch gdb

```
gdb-multiarch
(gdb) target remote localhost:1234
```

(gdb) file HelloWorld.elf // Type y to change the file if prompted 3
(gdb) load

Use gdb and the disassembly to answer the following questions-
On how to use gdb, see http://www.gnu.org/software/gdb/documentation/
On the ARM instruction set, see https://iitd-plos.github.io/col718/ref/arm-instructionset.pdf

1. What is the memory address of stack top? How much (in bytes) does the stack grow during the execution of the program?
2. At which memory location is the string “Hello world” stored? Verify using gdb.
3. How is this string passed to the function print uart0?
4. Identify the assembly instructions that implement the while loop of the function print uart0.
5. At which memory location is the address of the UART (i.e. UART0DR) stored? Verify using gdb.
6. What is the endianness of this processor?
7. Which registers are involved in implementing the C statement, *UART0DR = (unsigned int)(*s)