ARM Assembly Programming:

Part 1:

We started the assembly programming by opening a terminal, then opened a third.s file to write our program. We used the directive, .data, to indicate that we were going to declare some variables. We created a 2-byte signed memory size at location **a** initialized with -2. We then loaded the registers with some signed hexadecimal integers.

```
  @Third program
.section .data
a:.shalfword -2

.section .text
.globl _start
_start:

 mov r0,#0x1
 mov r1,#0xFFFFFFFF
 mov r2,#0xFF
 mov r3,#0x101
 mov r4,#0x400

 mov r7,#1
svc #0
.end
```

After we created the program, we tried to assemble it to get an object file and link, unfortunately, we couldn't assemble it because there was an error in the program with the following message:

```
pi@raspberrypi:~ $ nano third.s
pi@raspberrypi:~ $ as -g -o third.o third.s
third.s: Assembler messages:
third.s:3: Error: unknown pseudo-op: `.shalfword'
pi@raspberrypi:~ $
```

This error indicated that the 'shalfword' wasn't a keyword and therefore can't be used as a memory size. We fixed this error using the following program:

```
  @Third program
.section .data
a:.hword -2

.section .text
.globl _start
_start:

 mov r0,#0x1
 mov r1,#0xFFFFFFFF
 mov r2,#0xFF
 mov r3,#0x101
 mov r4,#0x400

 mov r7,#1
svc #0
.end
```

After the fix, we assembled and linked the program again, fortunately, it was able to assemble and link. We launched the gdb debugger for third.

We ran the **list** command to make sure our program was correct.

```
pi@raspberrypi:~ $ as -g -o third.o third.s
pi@raspberrypi:~ $ ld -o third third.o
pi@raspberrypi:~ $ gdb third
```

```
(gdb) list
1         @Third program
2       .section .data
3       a:.hword -2
4
5       .section .text
6       .globl _start
7       _start:
8
9        mov r0,#0x1
10       mov r1,#0xFFFFFFFF
(gdb)
```

Since we didn't load the variable **a** into any register, it didn't matter where we set the breakpoint as long as it is outside the **.data section**. We set a breakpoint at line 14 using the syntex **b 14** to avoid stepping an instruction each time since we will be showing the content of the registers too. After that, we ran the program using **run**. After the program was successfully executed, we checked the memory location to make sure that the integer initialized (-2) was present in the 2-byte sized memory. We used both **x/1xh** and **x/1xsh** accompaning with the address of **a** (**&a)** to display the content of the memory in hexadecimal.

```
(gdb) b 14
Breakpoint 1 at 0x10088: file third.s, line 15.
(gdb) run
Starting program: /home/pi/third

Breakpoint 1, _start () at third.s:15
15      mov r7,#1
(gdb) x/1xh &a
0x20090:        0xfffe
(gdb) x/1xsh &a
0x20090:        u"\xfffe"
```

We observed that viewing the content of the memory using the halfword (x/1xh) is different from the signed halfword (x/1xsh), however, they both seem to provide us with the 2-byte memory with the integer -2 in hexadecimal.

We also examine the content of the registers to make sure that everything that we loaded stayed in the registers.

```
(gdb) info registers
r0             0x1                 1
r1             0xffffffff          4294967295
r2             0xff                255
r3             0x101               257
r4             0x400               1024
r5             0x0                 0
r6             0x0                 0
r7             0x0                 0
r8             0x0                 0
r9             0x0                 0
r10            0x0                 0
r11            0x0                 0
r12            0x0                 0
sp             0x7efff3c0          0x7efff3c0
lr             0x0                 0
pc             0x10088             0x10088 <_start+20>
cpsr           0x10                16
fpscr          0x0                 0
```