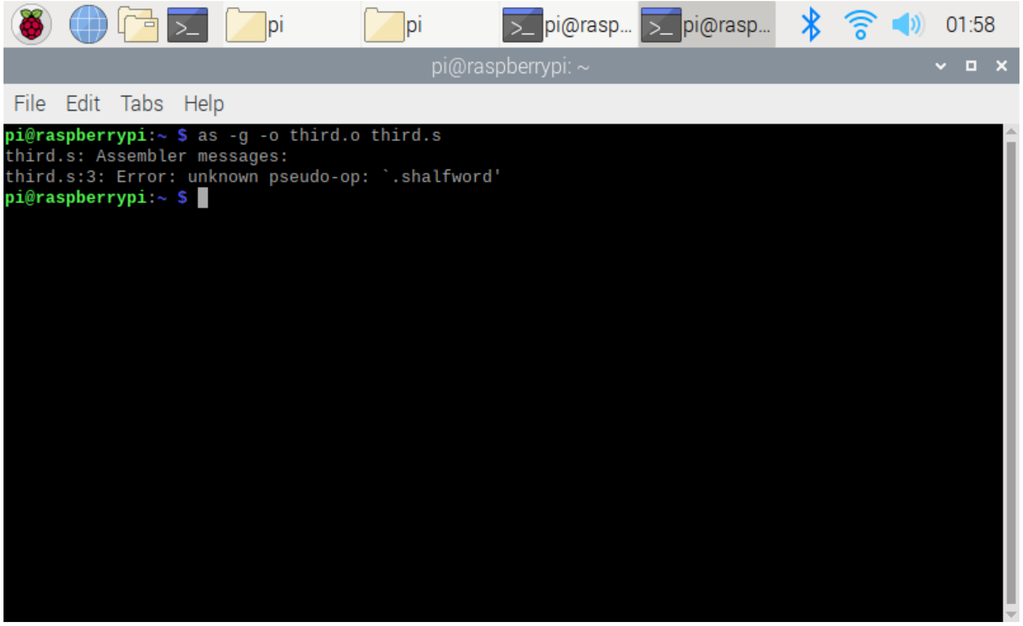
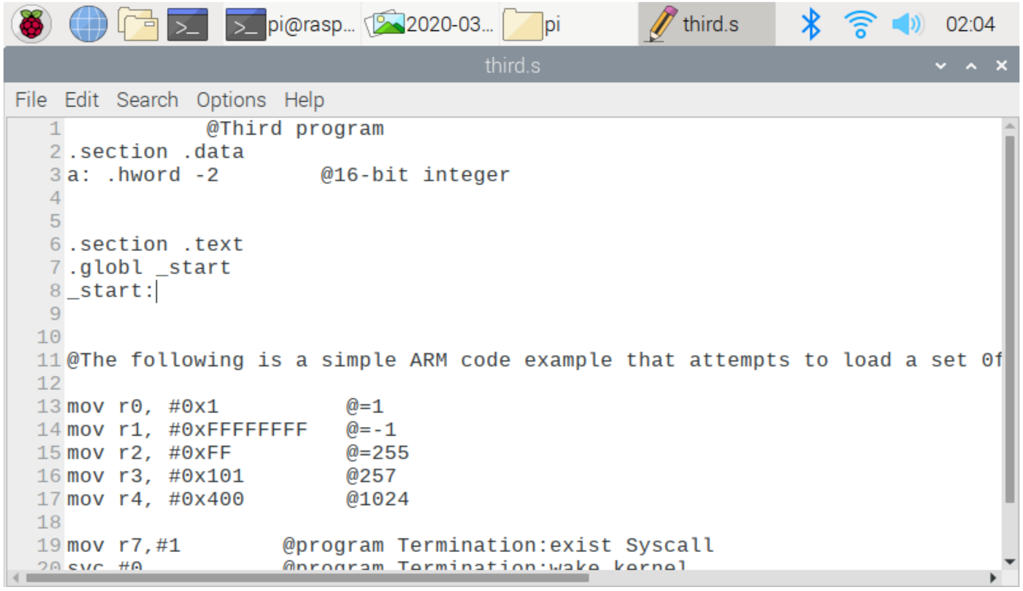
Arm Assembly Mezemir Gebre

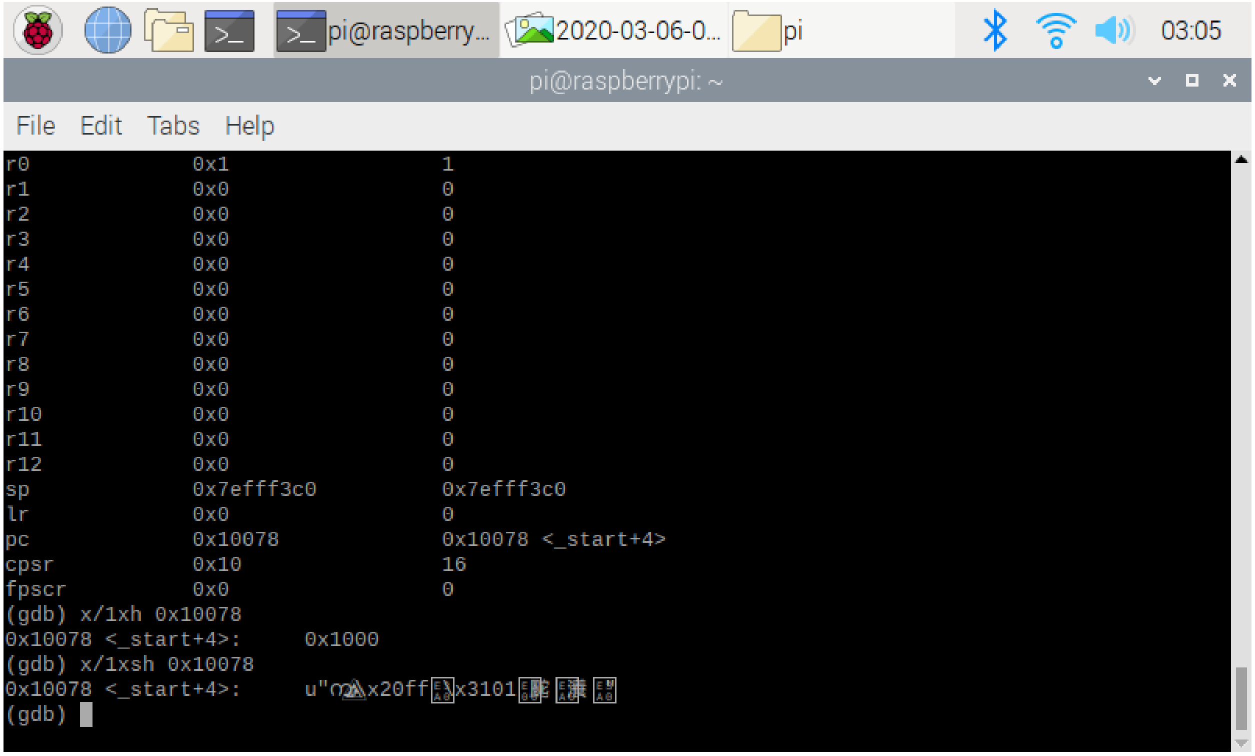
When I assembled the given program, it shows an error message with the data type shalfword -2. I find out that halfword in Arm assembly language comes as a signed data type by default, so we don’t need to specify it as a signed or put ‘s’ in front of it like in x86 assembly. To fix that I replaced it with “hword” and the program assembled correctly as shown in the second picture.



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

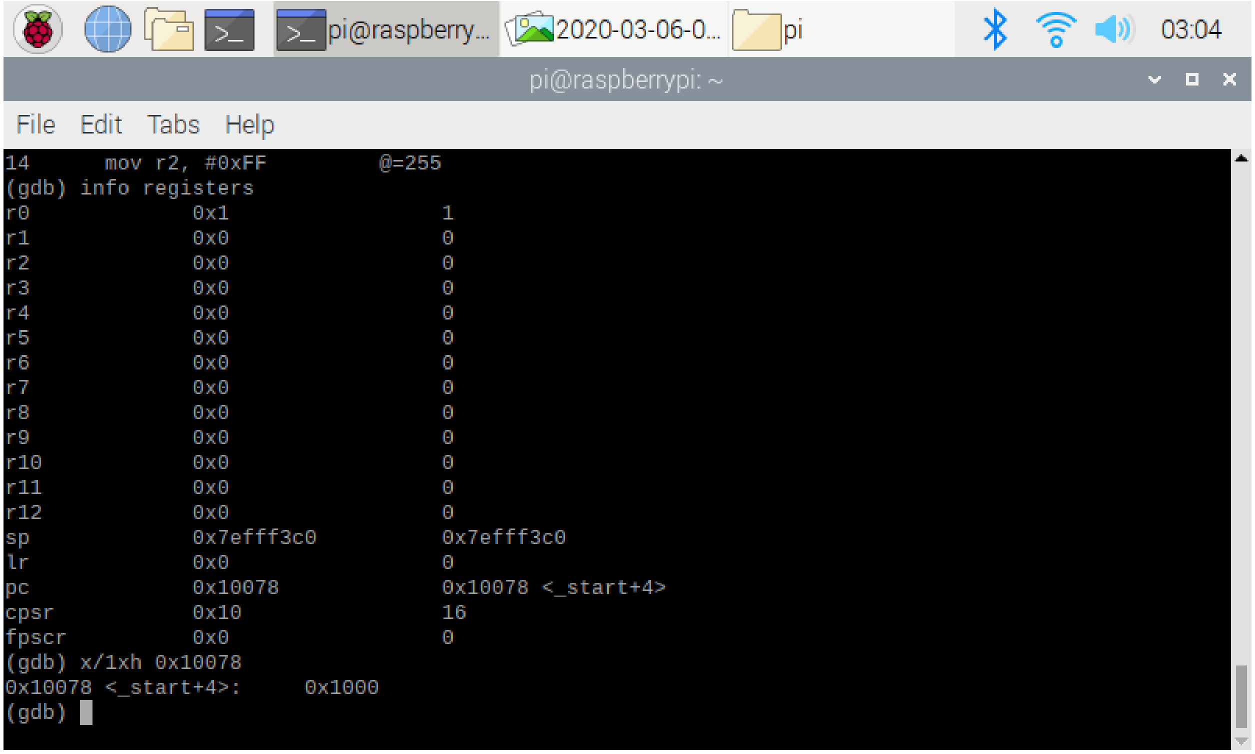


```
           @Third program
.section .data
a: .hword -2        @16-bit integer


.section .text
.globl _start
_start:


@The following is a simple ARM code example that attempts to load a set 0f

mov r0, #0x1          @=1
mov r1, #0xFFFFFFFF   @=-1
mov r2, #0xFF         @=255
mov r3, #0x101        @257
mov r4, #0x400        @1024

mov r7,#1        @program Termination:exist Syscall
svc #0           @program Termination:wake kernel
```



```
r0             0x1                 1
r1             0x0                 0
r2             0x0                 0
r3             0x0                 0
r4             0x0                 0
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             0x10078             0x10078 <_start+4>
cpsr           0x10                16
fpscr          0x0                 0
(gdb) x/1xh 0x10078
0x10078 <_start+4>:     0x1000
(gdb) x/1xsh 0x10078
0x10078 <_start+4>:     u"ന⚠x20ff[illegible]x3101[illegible]
(gdb) 
```

The above screenshot it the register info and memory access with ‘sh’ the value is not clear I think that implies that the data stored in the address ‘0x10078’ is unsigned but we trying to access a signed value in that address while there is unsigned value stored in it.

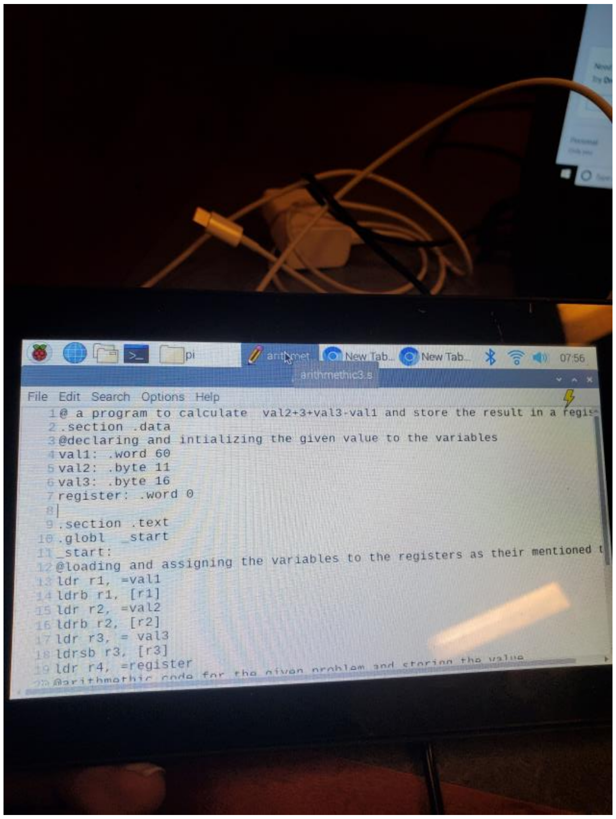


```
14      mov r2, #0xFF         @=255
(gdb) info registers
r0             0x1                 1
r1             0x0                 0
r2             0x0                 0
r3             0x0                 0
r4             0x0                 0
r5             0x0                 0
r6             0x0                 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             0x10078             0x10078 <_start+4>
cpsr           0x10                16
fpscr          0x0                 0
(gdb) x/1xh 0x10078
0x10078 <_start+4>:     0x1000
(gdb) 
```

The above screenshot is when I accessed the memory with h at address 0x10078 and the value in that address is displayed as hexadecimal number 0x1000.

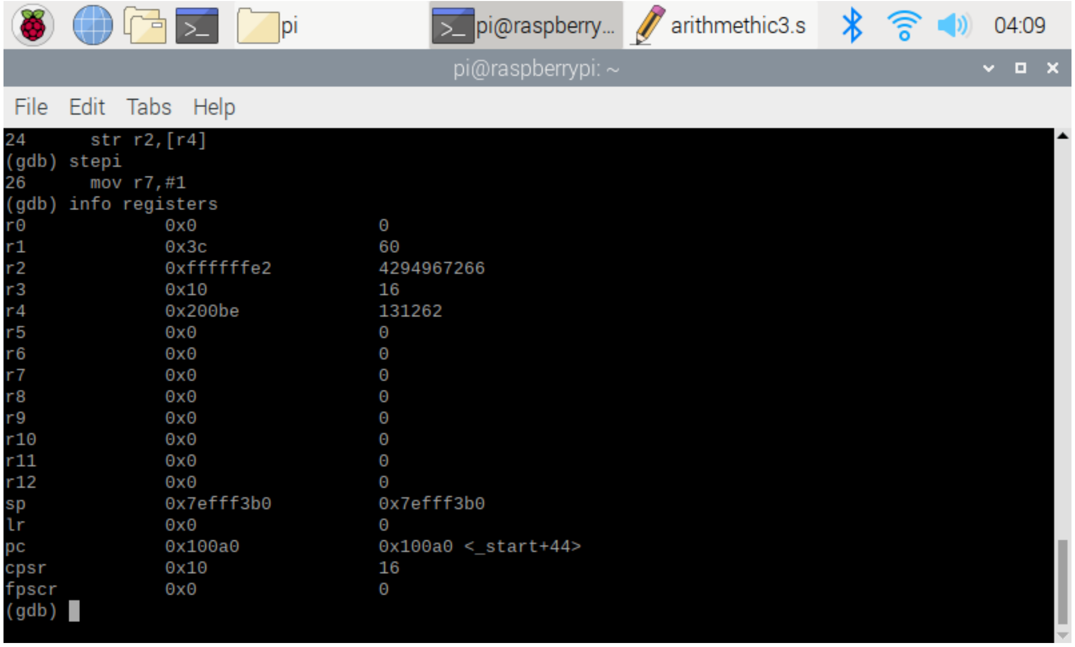
**Part 2**

The code for thearithmetic3 program where I was asked to write a program to do the following arithmetic (Register = val2 + 3 + val3 - val1) then I checked the result at the registers and the memory as the following screenshot shows. The data type declaration part was a bit challenging to me unlike the x86 assembly where we specify the data type by putting s in front of it if it is signed but in Arm assembly the data types comes as signed by default.



```
@ a program to calculate  val2+3+val3-val1 and store the result in a regis
.section .data
@declaring and intializing the given value to the variables
val1: .word 60
val2: .byte 11
val3: .byte 16
register: .word 0

.section .text
.globl  _start
_start:
@loading and assigning the variables to the registers as their mentioned t
ldr r1, =val1
ldrb r1, [r1]
ldr r2, =val2
ldrb r2, [r2]
ldr r3, = val3
ldrsb r3, [r3]
ldr r4, =register
@arithmethic code for the given problem and storing the value
```

The bottom picture shows the register values for the arithmetic3 program



```
24      str r2,[r4]
(gdb) stepi
26      mov r7,#1
(gdb) info registers
r0             0x0                 0
r1             0x3c                60
r2             0xffffffe2          4294967266
r3             0x10                16
r4             0x200be             131262
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             0x7efff3b0          0x7efff3b0
lr             0x0                 0
pc             0x100a0             0x100a0 <_start+44>
cpsr           0x10                16
fpscr          0x0                 0
(gdb) 
```

As you can see the result stored in the r2 register as “4294967266” and the ‘cpsr’ register indicates the signed value that are used in this program.