MCSL Lab03

ARM Assembly II

0310004
Kuan-Yen Chou

1. Purpose

- Get familiar with ARMv7 assembly language.

2. Steps

2.1 Postfix Arithmetic

Here is the code.

The functions in C POSIX library, like strlen and atoi, use zero as the end of a C-string. However, the strlen implementation in this lab uses both zero and 0x20, a whitespace, as the termination of a string, which may be a bad implementation but make the work easier. Namely, I don't have to implement strtok and to allocate spaces for the tokens. The atoi implementation then takes two arguments passed by registers. One is the beginning of the string, and the other is the length of the string returned by strlen. The atoi implementation is also in charge of handling the illegal input.

Therefore, strlen gets the next token length, and then isop checks whether it's an operator. If it is an operator, then do the computation. If it is not an operator, first try to convert the string to the corresponding integer (atoi), and then push the integer to stack. At last, move forward to the next token until the end, '\0'.

2.2 GCD and Stack Usage

Here is the code. I also implement a non-recursive version.

This exercise simply implements the Stein algorithm for computing the greatest common divisor of two given integers.

The GCD function first checks that if one of its arguments, passed by stack, is zero, then the other argument is to be returned. Then it checks whether or not its arguments are even. If both are even then recursively branch to itself, and the returned value, passed by r0, is doubled. If only one of its arguments is even, then just only recursively branch. If none of its arguments is even, then update the arguments accordingly, as in the both_odd procedure.

3. Results and Analysis

3.1 Postfix Arithmetic

Here is the execution result:

```
(gdb) load
Loading section .isr_vector, size 0x188 lma 0x8000000
Loading section .text, size 0x330 lma 0x8000190
Loading section .rodata, size 0x8 lma 0x80004c0
Loading section .init_array, size 0x8 lma 0x80004c8
Loading section .fini_array, size 0x8 lma 0x80004d0
Loading section .data, size 0x4b0 lma 0x80004d8
Start address 0x8000310, load size 2432
Transfer rate: 9 KB/sec, 405 bytes/write.
(gdb) display expr_result
1: expr_result = 0
(gdb) display *(int *)($sp + 8)
2: *(int *)($sp + 8) = 0
(gdb) display *(int *)($sp + 4)
3: *(int *)($sp + 4) = 0
(gdb) display *(int *)$sp
4: *(int *)$sp = 0
(gdb) b main
Breakpoint 1 at 0x80001b0: file src/main-1.s, line 15.
(gdb) b exam
Breakpoint 2 at 0x80001b2: file src/main-1.s, line 17.
(gdb) b forward
Breakpoint 3 at 0x80001c6: file src/main-1.s, line 25.
(gdb) b program_end
Breakpoint 4 at 0x80001da: file src/main-1.s, line 36.
(gdb) b compute
Breakpoint 5 at 0x80001dc: file src/main-1.s, line 42.
(gdb) b push_to_stack
Breakpoint 6 at 0x80001f8: file src/main-1.s, line 66.
(gdb) c
Continuing.
Note: automatically using hardware breakpoints for read-only addresses.

Breakpoint 1, main () at src/main-1.s:15
15          ldr   r0, =postfix_expr
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = 0
4: *(int *)$sp = 0
(gdb) c
Continuing.

Breakpoint 2, exam () at src/main-1.s:17
17          bl    strlen
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = 0
```

```
4: *(int *)$sp = 0
(gdb) c
Continuing.

Breakpoint 6, push_to_stack () at src/main-1.s:66
66          bl    atoi
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = 0
4: *(int *)$sp = 0
(gdb) c
Continuing.

Breakpoint 3, forward () at src/main-1.s:25
25          adds   r0, r0, r1
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = 0
4: *(int *)$sp = -100
(gdb) c
Continuing.

Breakpoint 6, push_to_stack () at src/main-1.s:66
66          bl    atoi
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = 0
4: *(int *)$sp = -100
(gdb) c
Continuing.

Breakpoint 3, forward () at src/main-1.s:25
25          adds   r0, r0, r1
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = -100
4: *(int *)$sp = 10
(gdb) c
Continuing.

Breakpoint 6, push_to_stack () at src/main-1.s:66
66          bl    atoi
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = -100
4: *(int *)$sp = 10
(gdb) c
Continuing.

Breakpoint 3, forward () at src/main-1.s:25
25          adds   r0, r0, r1
1: expr_result = 0
2: *(int *)($sp + 8) = -100
3: *(int *)($sp + 4) = 10
4: *(int *)$sp = 20
(gdb) c
Continuing.
```

```
Breakpoint 5, compute () at src/main-1.s:42
42          movs   r2, r3
1: expr_result = 0
2: *(int *)($sp + 8) = -100
3: *(int *)($sp + 4) = 10
4: *(int *)$sp = 20
(gdb) c
Continuing.

Breakpoint 3, forward () at src/main-1.s:25
25          adds   r0, r0, r1
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = -100
4: *(int *)$sp = 30
(gdb) c
Continuing.

Breakpoint 5, compute () at src/main-1.s:42
42          movs   r2, r3
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = -100
4: *(int *)$sp = 30
(gdb) c
Continuing.

Breakpoint 3, forward () at src/main-1.s:25
25          adds   r0, r0, r1
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = 0
4: *(int *)$sp = -130
(gdb) c
Continuing.

Breakpoint 6, push_to_stack () at src/main-1.s:66
66          bl     atoi
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = 0
4: *(int *)$sp = -130
(gdb) c
Continuing.

Breakpoint 3, forward () at src/main-1.s:25
25          adds   r0, r0, r1
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = -130
4: *(int *)$sp = 10
(gdb) c
Continuing.

Breakpoint 5, compute () at src/main-1.s:42
42          movs   r2, r3
1: expr_result = 0
2: *(int *)($sp + 8) = 0
```

```
3: *(int *)($sp + 4) = -130
4: *(int *)$sp = 10
(gdb) c
Continuing.

Breakpoint 3, forward () at src/main-1.s:25
25          adds   r0, r0, r1
1: expr_result = 0
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = 0
4: *(int *)$sp = -120
(gdb) c
Continuing.

Breakpoint 4, program_end () at src/main-1.s:36
36          b      program_end
1: expr_result = -120
2: *(int *)($sp + 8) = 0
3: *(int *)($sp + 4) = 0
4: *(int *)$sp = 0
(gdb) q
```

3.2 GCD and Stack Usage

Here is the execution result:

```
(gdb) load
Loading section .isr_vector, size 0x188 lma 0x8000000
Loading section .text, size 0x2a8 lma 0x8000188
Loading section .rodata, size 0x8 lma 0x8000430
Loading section .init_array, size 0x8 lma 0x8000438
Loading section .fini_array, size 0x8 lma 0x8000440
Loading section .data, size 0x430 lma 0x8000448
Start address 0x8000284, load size 2168
Transfer rate: 8 KB/sec, 361 bytes/write.
(gdb) display result
1: result = -2147419952
(gdb) display max_size
2: max_size = 251720120
(gdb) display *(int *)($sp + 8)
3: *(int *)($sp + 8) = 0
(gdb) display *(int *)($sp + 4)
4: *(int *)($sp + 4) = 0
(gdb) display *(int *)$sp
5: *(int *)$sp = 0
(gdb) b main
Breakpoint 1 at 0x8000190: file src/main-2.s, line 15.
(gdb) b GCD
Breakpoint 2 at 0x80001b2: file src/main-2.s, line 33.
(gdb) b GCD_l
Breakpoint 3 at 0x80001c2: file src/main-2.s, line 42.
(gdb) b loop
Breakpoint 4 at 0x80001b0: file src/main-2.s, line 30.
(gdb) c
Continuing.
```

```
Note: automatically using hardware breakpoints for read-only addresses.

Breakpoint 1, main () at src/main-2.s:15
15          mrs    r4, msp
1: result = 0
2: max_size = 0
3: *(int *)($sp + 8) = 0
4: *(int *)($sp + 4) = 0
5: *(int *)$sp = 0
(gdb)
Continuing.

Breakpoint 3, GCD_l () at src/main-2.s:42
42          mrs    r6, msp
1: result = 0
2: max_size = 0
3: *(int *)($sp + 8) = 1
4: *(int *)($sp + 4) = 133347
5: *(int *)$sp = 177796
(gdb)
Continuing.

Breakpoint 3, GCD_l () at src/main-2.s:42
42          mrs    r6, msp
1: result = 0
2: max_size = 0
3: *(int *)($sp + 8) = 1
4: *(int *)($sp + 4) = 133347
5: *(int *)$sp = 88898
(gdb)
Continuing.

Breakpoint 3, GCD_l () at src/main-2.s:42
42          mrs    r6, msp
1: result = 0
2: max_size = 0
3: *(int *)($sp + 8) = 1
4: *(int *)($sp + 4) = 133347
5: *(int *)$sp = 44449
(gdb)
Continuing.

Breakpoint 3, GCD_l () at src/main-2.s:42
42          mrs    r6, msp
1: result = 0
2: max_size = 0
3: *(int *)($sp + 8) = 1
4: *(int *)($sp + 4) = 44449
5: *(int *)$sp = 88898
(gdb)
Continuing.

Breakpoint 3, GCD_l () at src/main-2.s:42
42          mrs    r6, msp
1: result = 0
2: max_size = 0
3: *(int *)($sp + 8) = 1
4: *(int *)($sp + 4) = 44449
```

```
5: *(int *)$sp = 44449
(gdb)
Continuing.

Breakpoint 3, GCD_l () at src/main-2.s:42
42          mrs    r6, msp
1: result = 0
2: max_size = 0
3: *(int *)($sp + 8) = 1
4: *(int *)($sp + 4) = 44449
5: *(int *)$sp = 0
(gdb)
Continuing.

Breakpoint 4, loop () at src/main-2.s:30
30          b      loop
1: result = 44449
2: max_size = 12
3: *(int *)($sp + 8) = 0
4: *(int *)($sp + 4) = 0
5: *(int *)$sp = 0
(gdb) q
```

4. Reviews and Applications

It is my first time to write a little bigger program with assembly. I find that it's hard for me to stick to a nice coding style and to achieve good performance at the same time. However, since there is no scope concepts in assembly and the flow is single, so it is much easier to debug and to correct the wrong.