Report

190022658

February 24, 2021

1 Overview

In this practical, we were asked to analyse and document the assembly code with what the instruction do. There were three files to be analysed:

- array0-commented.s (unoptimised)
- array1-arm-commented.s (arm, optimisation lvl 1)
- array2-commented.s (optimisation lvl 2)

I analysed and commented all of them. Most of what they do is already commented in the assembly files. The rest are in this report.

2 Unoptimised Code

2.1 Summary

- In the unoptimised code, there is a lot of redundant copying. For example, there is no need to copy the arguments.
- Along with that, there are also 'and' instructions that are redundant.

2.2 Register use

Register	register	Space (in bytes)	value
-1 (%rbp)	%al	1	return value (boolean)
-8 (%rbp)		7	unused space
-16(%rbp)	%rdi	8	1st arg (array pointer)
-24(%rbp)	%rsi	8	2nd arg (elem)
-28(%rbp)	%edx	4	3nd arg (left)
-32(%rbp)	%ecx	4	4nd arg (right)

Here, -8(%rbp) to -1(rbp) remains unused throughout, while -36(%rbp) to -48(%rbp) gets used sometimes. 4 bytes in registers rdx and rcx are also not used (but they are not accessed either). And there are a lot of duplicates.

3 Optimised Code

3.1 Summary

- In the optimised code, there is almost no redundant copying. There's no copying of the arguments
- Here, there are no function call. The original recursive function has been converted into an iterative form.
- This code has some nice parts like where in the case where elem $<$ array[mid], instead of starting again by simply assigning the ecx register the value of mid it does mid - left (mid is the new left) and continues. This is a very small bit but goes on to show that even here a redundant copying was not done (which would have been the case if the code would have started again).
- The way to divide by 2 is quite interesting. The bits that had to be divided were pushed to the right by one byte. This divides the bits. This was done with (right - left).

```
10  (2) -> 01  (1)
100 (4) -> 010 (2)
110 (6) -> 011 (3)
```

3.2 Register use

Register	Space (in bytes)	Value
%al	1	return value (boolean)
%rdi	8	1st arg (array pointer)
%rsi	8	2nd arg (elem)
%edx	4	3nd arg (left)
%ecx	4	4nd arg (right)
%r8d	4	helper

Here, rax (1 byte used), rdx (4 byte used), rcx (4 byte used) are not used completely, but (again) are not accessed completely either.

4 ARM Code

4.1 Summary

Here, I learn't about addressing modes, registers, and instructions used in ARM64. There were places which did not make sense like the cinc instruction that is never true, and never increments the value of w8.

`https://modexp.wordpress.com/2018/10/30/arm64-assembly/`
This blog is where I learn't the most from.

4.2 Addressing modes

- [base]
- [base, offset]
- [base, offset]! -> this means adding the offset after the instruction
- [base], offset -> here the base also changes
- [base, register, sxtw #3] -> here the register is 32-bit thus the need to use sxtw

4.3 Registers

x29 is frame pointer

x30 is link register

x0-x7 arguments and return value

x8-x18 temporary registers

sp stack pointer

4.4 Instructions

- Here, the instructions generally take a destination register too.

For example -

```
sub w8, w3, w2      ;; w8 = w3 - w2
```

- The add instruction

```
;; w8 = (right-left)
;; arithmetic shift right to w8
;; w8 = (right-left)/2
add w8, w2, w8, asr #1      ;; w8 = left + (right-left)/2
```

4.5 Register use

Register	Space (in bytes)	Value
w0	4	return value (boolean)
x1	8	1st arg (array pointer)
x2	8	2nd arg (elem)
w3	4	3nd arg (left)
w4	4	4nd arg (right)
x8	8	helper
x9	8	helper
w8	4	helper

5 Conclusion

This practical was a good self-learning experience. This also showed that a lot of knowledge is transferable.

6 Bibliography

- https://modexp.wordpress.com/2018/10/30/arm64-assembly/
- https://web.stanford.edu/class/cs107/guide/x86-64.html
- https://stackoverflow.com/questions/17170388/trying-to-understand-the-assembly-in (cltd)
- https://stackoverflow.com/questions/37743476/assembly-cltq-and-movslq-difference (clqt is equivalent to movslq %eax %rax)
- https://stackoverflow.com/questions/9317922/what-does-the-movzbl-instruction-do-i