Prof. Dr.-Ing. Tim Güneysu
M. Sc. Florian Stolz
M. Sc. Moritz Peters

Ruhr-Universität Bochum
Faculty of Computer Science
Chair for Security Engineering

Winter Term 23/24

Computer Engineering 3 – Hardware-Oriented Programming

Project #2

Submit answers until 16. November 23:59:59 via *Moodle* Σ 100 Points

---

Attention: No external library functions are allowed! All functions are located in `tasks/` and tests are provided in `main.c` . Comments may provide you with further instructions. Ask us if something is unclear.
Attention: Comment your code! Code that is not commented, not understandable and not working will potentially get less points. Comments only have to describe the overall idea, you do not have to comment each instruction.

Task 1: Hashing (25 Points)

Hash tables are a great way to organize data. Instead of having a simple array and looking up where the data is we are looking for (e.g., by looping through all indexes and comparing an identifier), we *hash* an identifier which gives us a unique number, which acts like an index. Then, we simply have to access the index to get our data. In order to perform this task relatively fast, we implement it in ARM Thumb2 Assembly.

a) Implement the function `uint32_t hash(uint8_t* k, uint32_t length)` in `task1.c`, which loops over all bytes in `k`. The initial hash is 0. In each iteration `hashop(uint32_t hash,uint_t* k, uint32_t index)` is called, which performs the actual hash. (12.5 Points)
b) Implement the function `uint32_t hashop(uint32_t hash,uint_t* k, uint32_t index)` in `task1aux.s`, which performs the actual hashing operation on the selected byte index. It gets the current hash value, the pointer to the byte array and the index. It returns the updated hash. The hashing operation works as follows: Multiply the current hash by 33 and then add the byte to it. (12.5 Points)

Task 2: Caesar In-Place (25 Points)

Remember the crazy good encryption scheme you learned in Tutorial 2. Now it is time to implement it in ARM Thumb2 Assembly. The Encryption works as follows: It *only* works on lowercase letters and adds a key to it. For example, if the key is 3 then 'a' becomes 'd'. You only need to implement the encryption! Here is the catch: Do not simply add the key to 'z' and output whatever comes after 'z'. Instead, loop back to the beginning of the alphabet. So if the key is 3, then 'z' becomes 'c'. Furthermore, the output should overwrite the input.

a) Implement the function `void encrypt(char* str, int len, int key)` in `task2.c`, which loops over all chars in `str`. In each iteration `encryptop(char* letter,int k` is called, which performs the actual encryption operation. (12.5 Points)
b) Implement the function `void encryptop(char* letter,int k)` in `task2aux.s`, which performs the actual encryption operation on the selected letter. It gets a pointer to the current letter and the key. The encryption function is described above! (12.5 Points)

Task 3: Financial Advice (50 Points)

In such turbulent times it is always nice to have sound finances, especially if you are as large as LAME Inc. The *Net Present Value* can give you an idea, whether an investment performed now will actually give you some profit later on. This works by summing up all inflows you get from the given investment *discounted* by the interest you would have gained on the free market and then subtracting the initial investment amount. If the result is < 0, your investment actually loses you money. If the result is 0, you gain nothing. However, if the result is > 0, you make a profit.
Example: You lend 3000 Euros to Maike, who promises you to give you 1000 Euros each year for 3 years. The interest rate on the free market is 5%. Thus: $-3000 + \frac{1000}{1.05^1} + \frac{1000}{1.05^2} + \frac{1000}{1.05^3} = -276.75$. Therefore, lending her money on these conditions is clearly unprofitable for you. However, if she promises to give you 1200, 1100 and lastly 1000 euros then: $-3000 + \frac{1200}{1.05^1} + \frac{1100}{1.05^2} + \frac{1000}{1.05^3} = 4.42$. You gain 4 Euros.

The formula is: $c = -initial + \sum_{n=0}^{t} \frac{income_n}{(1 + interest)^n}$

a) Implement the function `float npv(float init, float* income, int years, float interest)` in `task3.s`. (50 Points)

Hint: In order to debug floating-point operations, use the `vmov rX, sX` instruction and set the number format to `Float` in the debugger for your register `rX`!
Hint: If you want to move values between floating-point registers, you need to use the `vmov.f32 sX, sY` instruction!