Assignment 1 2019 Fall

Texas A&M University Computer Science

X86 Assembly Basics

Background

Assembly is at the core of every computer language and is the last phase before transitioning to machine encoding of 1's and 0's. Creating programs for video games, web interfaces, and other modern applications would not be feasible to develop or maintain in assembly, because simple operations require knowledge of the architectures machine code instructions (registers, opcodes, flags, and other low level details). Thus, modern languages such as Python, Perl, C (including all variations of C), and Java have created an abstraction to assembly that allows programmers to design and implement software that is much easier to maintain and create. Each of these languages may use different compilers or interpreters that change the flow of the program for different types of optimizations, but will still be translated into basic assembly (symbolic machine code) by an assembler. The code generated by the assembler is much larger than the original program and much harder for humans to understand. This makes the reverse engineering process to discover secrets, behavior patterns, protocols, vulnerabilities, and exploits much more difficult, because the original source code and compiler is generally unknown. Code may also be manipulated by obfuscators, which is a technique used to make it harder to understand the intended source or machine code. Thus, understanding how basic operations such as *adding*, *multiply*, *division*, *push*, *pop*, and *move* are critical to understanding the structure, function, and operation of the program.

Assembly Registers

Many of you have probably implemented a stack in Python or c++, computed averages, specific min/max of numbers in a set, range of sequences, loops, multiplication, and sums, so this assignment should be intuitively simple. However, when you added two numbers in Python you did not have to keep track of what register the value should go into. X86 Registers are the main tools to writing any program in assembly, also known as process registers (listed below). The ones listed in yellow will be the focus of this assignment. The ESP or SS registers are reserved for implementing a stack. Remember that the top of a stack is the last inserted item into the stack and is pointed by the SS registers. The ESP gives the remaining offset of the stack.

General Registers	Segment Registers	Index and Pointers	Indicator
EAX	CS	ESI	EFLAGS
EBX	DS	EDI	
ECX	ES	EBP	
EDX	FS	EIP	
	GS	ESP	
	SS		

*Read sections **1.3.1, 1.4.1, 1.5.1, and 1.7** in the Reverse Engineering for Beginners book*

Problem 1 (10 points)

A stack is similar to an array structure, in that data that is added or removed from the structure is pushed and popped from the top of the stack (LIFO data structure). Thus, assembly provides two operands for preforming this task:

1) PUSH operand
2) POP address/register

An example of this is found on tutorialspoint website.

Task: The goal of this problem is to complete the following:

1) Push the integers 4, 77, 18, 57, and 9 onto the stack.
2) Take the average of these numbers by popping them from the stack one at a time and storing the result into the EAX register.

Problem 2 (20 points)

The results of problem 1 will be used to complete the below task. That is, the average of 4,77,18,57, and 9 that is currently stored in the EAX register.

Task: Complete the following:

1) Create a sequence of numbers from the average result of problem 1 task 2, that is, a set of numbers from 1 to the current EAX value.
2) Compute the average of this set and store the result in the EAX register.
3) Computer the average again, but this time ignore numbers that are between 20 and 30.
4) Lastly, compute the range of the set (sequence of numbers) and store this into the EDX register. A range of a set of numbers is the highest number minus the lowest number.

Problem 3 (25 points)

Create a program that can loop through a sequence of numbers that may include duplicated values.

Task: Complete the following:

1) Push the numbers 0 to 100 onto the stack, however, when the current value is equal to 42, include nine additional 42 values. That is, the resulting list should be the following:

   Sequence of numbers: {0,1,…,42,42,42,42,42,42,42,42,42,42,…,100}

2) Compute the number of iterations in the loop and store this value in the EAX register. You may want to use the add operand as you create the sequence of numbers.
3) After the sequence of numbers is built, loop through them and compute the product of any numbers that are less than 30 and store this into the EAX register. That is, multiply each number together < 30
4) Lastly, compute the sum of the sequence of numbers and store this in any register of your choice.

Report (45 points)

Summary of the assignment, what you learned, and how you this might help you reverse engineer code in the future. Any critiques can be given on the assignment, but must be followed with an answer on how to change the assignment. This must be one page **11 point TIMES NEW ROMAN** font, **or you will lose points.**

BONUS: Any of the following task are optional.

1) Create a program in assembly that can do one or all of the following:
    a. Print a message onto the screen
    b. Open a window
    c. Run a program
2) Create any type of encryption algorithm of your choice in assembly. You can make up your own hash function. Store the original unencrypted value in EAX and the resulting value into EBX. Additional credit will be given for decrypting the value and comparing it to the original value.

TOTAL POSSIBLE POINTS

Problem 1	Problem 2	Problem 3	Report	Bonus
10	20	25	45	Up to 25 points

WHAT TO HAND IN

Provide a word document with the assembly code **NEETLY** labeled. Basically, do not do something that is hard to read or **you will lose points**. The word document will contain 4 pages (5 or more if the bonus is completed.) DO NOT COPY PASTE SOLUTIONS FROM ONLINE. Google is extremely easy to navigate and any solutions with poor explanations are obvious signs that the student may have cheated.

Page 1

One-page report

Page 2

Problem 1 Assembly code and instructions on how you ran it

Page 3

Problem 2 Assembly and instructions on how you ran it

Page 4

Problem 3 Assembly and instructions on how you ran it

Page 5+

Bonus. This will include a paragraph or more summary on what you did IF you chose to #2 (encryption)