Lab Tasks

You can use the AUCA server for lab work through SSH at 'auca.space'. The login has the format `<last name>_<first letter of the first name>`.

There are multiple ways to prepare a working environment on Windows ranging from using a virtual machines running Linux, using the Linux Subsystem on Windows, using the MinGW and MSYS development environment, or the Cygwin Unix emulation system among others.

In all cases ensure that you have the following packages installed

- coreutils
- binutils
- gcc
- make
- gdb

Lab #1, Basics

Task #1, messages with C

1. Create a C application that prints some text message to the standard output stream.
2. Debug the program with GDB. Use the text UI mode of the debugger.

```
gdb -tui ./<Executable File Name>
```

3. Try to preprocess, generate assembly, compile, and link the program step by step.

```
gcc -E -o <Preprocessed File Name> <Source File Name>
gcc -S -o <Translated Assembly File Name> <Source File Name>
gcc -c -o <Object File Name>.o <Source File Name>
gcc -o <Executable File Name> <Object File Name>
```

3. Disassemble the program with objdump and send the output to the pager program `less`. Type `/main<ENTER>` to search for the `main` function. Type `n` or `N` to go forward or backward between your search results. Navigate up and down with `j` and `k` keys. Type `q` to exit from the pager.

```
objdump -D <Executable File Name> | less
```

Task #2, messages with the x86-64 assembly

1. Write an x86-64 assembly program to print some message to the standard output stream. Use the `puts` C standard library function to output the text message to the screen. Use the GNU Assembler to translate your program into the object file.
2. Step through the instructions with GDB.
3. Rewrite the program in x86 assembly for the 32-bit architecture.

4. Rewrite one program by using the Intel assembly syntax.

Task #3, messages directly through the operating system

1. Rewrite the 32-bit and the 64-bit programs from task #2 without using the `puts` function call. Call the operating system kernel directly. Use the `write` system call.

   man 2 write

Task #4, increment and decrement

1. Write a program that reads one integer number from the standard input stream, increments the number by one, and writes the number to the standard output stream.
2. Write a program that reads one integer number from the standard input stream, decrements the number by one, and writes the number to the standard output stream.

In both cases use the C standrad library functions `scanf` and `printf`. In both cases use the the x86-64 assembly.

```
man 3 scanf
man 3 printf
```

Task #5, Sum of two numbers

- Write a program in x86-64 assembly that reads two numbers from the standard input stream, sums them together, and outputs the result to the standard output stream.

Task #6

- Write a program in x86-64 assembly to find a sum of 10 integer numbers. Create a function to sum the numbers.

Task #7

- Read two numbers and find if the first number is greater than the second one, the second is greater than the first one, or both numbers are equal. Print an appropriate message.

Task #8

- Write a program with a recursive function to find an `n`'th Fibonacci number.

Task #9

- Write a program with a recursive function to find a GCD of two numbers.

Task #10

- Write a program with a recursive function to find a factorial of `n`.

Task #11

- Create non-recursive versions of tasks #8–10.

Task #12

- Create functions in C to calculate the number of days in a month for a given year and month number. Write a version with an `if` construct, a version with a `swith/case` construct, and a version with a lookup array for values.
- Write the version with a lookup array with values in assembly from scratch.

Task #13

- Write a program that given an array `input` of 30 numbers (any numbers) in code, finds and writes the result of `input[i] * a + b` back to the array and then prints the resulting array. The numbers `a` and `b` should be read from the standard input with `scanf`.

Homework

Finish all the tasks to prepare for the exams. Rewrite all the programs from tasks #4, #5, and #8 in x86 32-bit assembly.

Compilation

To compile your C program on Linux run the following

```
CFLAGS=-g make <Source File Name>
```

or

```
gcc -g -o <Name of the Executable> <C Source File Name>
```

To translate and link your assembly program run the folowing

```
gcc -o <Name of the Executable> <Assembly Source File Name>
```

To start the executable

```
./<Name of the Executable>
```

To compile a C program with optimizations run one command from the following list depending on the level of optimization that you need (`O1-3` or `Os` for size).

```
gcc -O1 -o <Name of the Executable> <C Source File Name>
gcc -O2 -o <Name of the Executable> <C Source File Name>
gcc -O3 -o <Name of the Executable> <C Source File Name>
gcc -Os -o <Name of the Executable> <C Source File Name>
```

Documentation

```
man make
man gcc
man as
man gdb
man objdump
```

Links

C

- Beej's Guide to C Programming

GDB

- GDB Quick Reference

Radare2

- Radare2 Visual Graphs

x86 ISA

- Intel® 64 and IA-32 Architectures Software Developer Manuals
- System V AMD64 ABI
- X86 Opcode Reference
- X86 Instruction Reference
- Optimizing Subroutines in Assembly Language
- Jump Quick Reference

Assemblers

- Linux assemblers: A comparison of GAS and NASM
- GAS Syntax

Books

C

- C Programming: A Modern Approach, 2nd Edition by K. N. King

x86 Assembly

- Assembly Language for x86 Processors, 7th Edition by Kip R. Irvine