Parallel Programing
Assignment 1

deadline: 3.6.2024 23:59

Submission Guidelines

To reduce likelihood of misunderstandings, please follow these guidelines:

1. Work can either be done individually or in pairs.
2. Submission is through the “submit” system.
3. Make sure that your solution compiles and runs without any errors and warnings on BIU servers.
4. In the first line of every file you submit, write in a comment your id and full name. For example: “/* 123456789 Israela Israeli */”.
5. Not using SSE/AVX (either as pure assembly, or as intrinsics, according to the instructions of the task) in even one task will result in an automatic 0 for the grade. In other words, **sequential code is unacceptable.**

General Background

The goal of this exercise is to practice assembly using SSE/AVX instructions,
and C SSE/AVX intrinsics. This means that you'll be writing both assembly and C code. Every task will specify which one you should use.

This exercise is composed of 2 unrelated parts. The first part is about string processing and the second is about floating point calculations. Each part will consist of several different tasks, and use different source files for its environment.

Environment & Submission

As we stated before, each part of this assignment uses different source files. The sources for part 1 can be found inside the **strings** directory and for part 2 inside the **formulas** directory. For each one of the parts, we'll specify exactly what files you need to write and how, but keep in mind that overall you need to submit two zip files, one for each assignment, named and containing the following:

1. **"strings.zip":** hamming.s, b64.c and Makefile
2. **"formulas.zip":** formula1.c, formula2.s and Makefile

We give you the freedom to provide your own makefiles, **just make sure your programs compile and run correctly on BIU servers.**

Part I - Strings

In this part, we'll implement two string operators using SSE/AVX instructions and intrinsics. For this part, open the **strings** folder using your text editor of choice. As you can see, the main runs a loop that receives two strings from the user, then prints the result of some operation on them. You'll be the one implementing them.

Hamming Distance

int hamming_dist(char str1[MAX_STR], char str2[MAX_STR]);

The first operation is the classic *hamming distance*. The hamming distance of two strings is the number of positions at which the corresponding symbols are different. Write the implementation of the function in a file named **"hamming.s"**, using Assembly only.
(note- if the strings are not in the same length, the difference of the length should be added as a hamming distance as well).

Base 64 Distance

int b64_distance(char str1[MAX_STR], char str2[MAX_STR]);

The second operation you'll need to implement is base 64 distance. We define the base64 distance of two strings in the following way:

1. Take the two given strings and remove any non-base 64-characters (i.e a-z,A-z,0-9,+,/) while maintaining the order of every other character.
2. Calculate the numbers represented by the strings, and subtract the first from the second.
3. Return the results.

Write the implementation of the function in a file named **"b64.c"**, using Intrinsics.

Part II - formulas

For this part, open the **formulas** folder. Here, the main runs the functions that are defined in the file "formulas.h" on some test values, and compares their results to the correct results for these examples. Because we are dealing with float/double variables, there might be rounding errors, so to compare your results with the correct one, we will use the comparison function is_close that is written in the "main.c" file.

The First Formula

float formula1(float *x, unsigned int length);

Given an array of floating point numbers, x, and its length, the function needs to calculate and return the following expression:

$$\sqrt{1 + \frac{\sqrt[3]{\sum_{k=1}^{n} \sqrt{x_k}}}{\prod_{k=1}^{n} ({x_k}^2 + 1)}}$$

Write the implementation of the function in a file named **"forumla1.c"**, using Intrinsics.

The Second Formula

float formula2(float *x, float *y, unsigned int length);

x and y will be arrays of floats of the given length. The function needs to calculate and return

$$\sum_{k=1}^{n} \frac{x_k y_k}{\prod_{i=1}^{n} ({x_i}^2 + {y_i}^2 - 2x_i y_i + 1)}$$

Write the implementation of the function in a file named **"formula2.s"**, using Assembly.

Notes

- You can assume that the input is divisible by 4.
- The maximum length of the strings will be 255 characters (and one more character for the nullbyte). The strings cannot be assumed to be the same length.
- You can use the code of strlen that we showed you in the Tirgul.
-
- For your convenience and for a better understanding, here is Python code that converts a string from base 64 to a number in base 10 (where the code completely ignores characters that are not part of base 64) in the way we described in the question:

```Python
def b64(string):
    x = 0
    for c in string:
        if 'A' <= c and c <= 'Z':
            x = 64 * x + ord(c) - ord('A')
        elif 'a' <= c and c <= 'z':
            x = 64 * x + ord(c) - ord('a') + 26
        elif '0' <= c and c <= '9':
            x = 64 * x + ord(c) - ord('0') + 52
        elif c == '+':
            x = 64 * x + 62
        elif c == '/':
            x = 64 * x + 63
    return x
```