Parallel Programing - Assignment 1

Objective

In this assignment, you will implement 4 tasks that utilize SIMD capabilities, both through assembly and intrinsics. The focus is on performance optimization using SSE/SSE4.2 instructions.

Task 1: Decode Function Implementation - warm up

Background:

For a function:

```
long decode(long x, long y, long z);
```

The following assembly code is generated:

```
decode:
    subq %rdx, %rsi
    imulq %rsi, %rdi
    movq %rsi, %rax
    salq $63, %rax
    sarq $63, %rax
    xorq %rdi, %rax
    ret
```

Parameters x, y, and z are passed in registers %rdi, %rsi, and %rdx.
The return value is stored in %rax.

Requirements

Implement decode in C with the same effect as the given assembly code.

Save your implementation in a file named: " decode.c"

Declare the func `long decode_c_version(long x, long y, long z);`

Task 2: String Length with SSE

Background:

The provided _strlen function calculates the length of a string in assembly:

```
_strlen:
    pushq %rcx              # Save rcx on stack
    xorl %ecx, %ecx         # Zero counter (faster than `xorq %rcx, %rcx`)

_strlen_next:

    cmpb $0, (%rdi)         # Compare byte at rdi with 0 (null terminator)
    je _strlen_done         # If zero, exit loop
    incq %rcx               # Increment character count
    incq %rdi               # Move to the next character
    jmp _strlen_next        # Repeat

_strlen_done:
    movq %rcx, %rax         # Move counter to return value (rax)
    popq %rcx               # Restore rcx
    ret                     # Return to caller
```

Requirements:

Implement `strlen_sse42`

Implement an optimized version of the strlen function using **SIMD instructions** (specifically SSE) in **x86-64 assembly**.

Note:

In the lectures, many assembly instructions may have appeared using **both Intel and AT&T syntax**.

As part of this assignment, **students are expected to research and understand the correct syntax**, operand order, and usage of instructions like `pcmpistri`, `pcmpistrm`, and other SIMD operations.

discovering how to use these instructions properly—especially their behavior, operand rules, and control byte configuration (imm8)—is a core part of the learning process.

Instructions like pcmpistri and pcmpistrm require careful setup. Research their operand usage and control byte (imm8). Example AT&T syntax:

```
    pcmpistri $imm8, %xmm, %xmm
```

Task 3: Hamming Distance

In this part, we'll implement two string operators using intrinsics.

Open the strings folder using your text editor of choice. The main function runs a function that receives two strings and prints the result of some operation on them. Your task is to implement the Hamming Distance function using intrinsics in C.

Hamming Distance

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

Definition:

The Hamming Distance of two strings is the number of positions where the corresponding characters are different.

Requirements

- Implement the function using intrinsics.
- If the strings are not the same length, the difference in their lengths should be added to the final Hamming distance.

Implementation Details

- Use intrinsics to compare multiple characters at a time, instead of checking one character at a time in a loop.
- Store the comparison results in a vector register and count the differences efficiently.

Input

```
str1 = "hello world"
str2 = "hxllo worl!"
```

Output

```
Hamming Distance = 2
```

Write the implementation of the function in a file named hamming_intrinsics.c

Task 4: The Formula

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.

```
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:

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

Write the implementation of the function in a file named “forumla1.c”, using Intrinsics.

Submission guidelines

1. Work can either be done individually or in pairs.
2. Submission is through the moodle (lamda).
3. Ensure your code compiles and runs without errors or 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. Failure to use SSE/AVX (intrinsics or pure assembly) where required will result in an automatic 0.
6. **The zip ex1.zip should include the following files:**

```
ex1.zip
 ├ decode.c
 ├ strlen_sse42.s
 ├ hamming_intrinsics.c
 ├ formula1.c
```