# CSD1100 Programming Assignment: Lab 10 (Assembler - Basics)

## Topics and references

- Assembler programming basics
- Registers, function parameters and return value
- Arithmetic instructions
- pop and push

## Task

In this assignment you have to implement a few functions that take parameters and return result of calculation of an equation with constant values and basic arithmetic operators.

Complete instruction to the assignment and explanation of how to use assembler instructions `add`, `sub`, `imul`, `idiv` **will be given in the class**.

You implementation must passes all given tests in order to get the full mark for the assignment.

## Submission details

Please read the following details carefully and adhere to all requirements to avoid unnecessary deductions.

### Source files

You have to submit your implementation of the functions in the source file `functions.asm`.

```
; ------------------------------------------------------------------------
; File: functions.asm
; Project: CSD1100 Assignment 10
; Author: Vadim Surov, vsurov@digipen.edu
; Co-Author: Your name, email
;
; Compile: nasm -f elf64 functions.asm -o functions.o
; Link: gcc main.o functions.o -o main.o -lm
;
; Copyright: 2021, Digipen Institute of Technology, Singapore
;
; Note: All functions use at most 6 parameters
;       p1, p2, p3, p4, p5, p6
;       located in registers
;       rdi, rsi, rdx, rcx, r8, r9
;       accordingly.
; ------------------------------------------------------------------------

    section .text

    global f1
```

```
    global f2
    global f3
    global f4
    global f5

f1:
; TODO: Return 5th integer parameter.

    ret

f2:
; TODO: Return the result of addition of the first 3 integer parameters.

    ret

f3:
; TODO: Return the result of calculation p1 * p2 + p3 / p4 + 1,
;       where operator / calculates an integer result.
; Tip: Be aware that parameter p4 in rdx is used in both * and /.

    ; r15 = p3 / p4

    ; rax = p1 * p2

    ; rax += r15

    ; rax++

    ret

f4:
; TODO: Return the result of calculation
;       p1 * 100000 + p2 * 10000 + p3 * 1000 + p4 * 100 + p5 * 10 + p6 * 1.
; Tip: Use push/pop to save rdx temporarily in stack to do multiplications.

    ; r11 = p1 * 100000

    ; r12 = p2 * 10000

    ; r13 = p3 * 1000

    ; r14 = p4 * 100

    ; r15 = p5 * 10

    ; rax = p6 + r11 + r12 + r13 + r14 + r15

    ret

f5:
; TODO: Return the result of calculation
;       p1 / 100000 - p2 / 10000 - p3 / 1000 - p4 / 100 - p5 / 10.

    ; r11 = p1 / 100000

    ; r12 = p2 / 10000

    ; r13 = p3 / 1000
```

```
    ; r14 = p4 / 100

    ; r15 = p5 / 10

    ; rax = r11 - r12 - r13 - r14 - r15

    ret
```

Test cases are given in `main.c` file. Do not change it. It won't be submitted anyway.

```
/*-----------------------------------------------------------------
  File: main.c
  Project: CSD1100 Assignment 10
  Author: Vadim Surov, vsurov@digipen.edu

  Compile: gcc -c -Wall -Wextra -Werror main.c -o main.o
  Link: gcc main.o functions.o -o main.o -lm

  Copyright: 2021, Digipen Institute of Technology, Singapore
-----------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>

// See the function description in functions.asm
int f1();
int f2();
int f3();
int f4();
int f5();

void test1();
void test2();
void test3();
void test4();
void test5();
void test6();
void test7();
void test8();
void test9();
void test10();

int main(int argc, char* argv[])
{

    void (*f[])() = { test1, test2, test3, test4, test5, test6, test7, test8,
test9, test10 };
    const int SIZE = sizeof(f) / sizeof(f[0]);
    int id = -1;

    if (argc == 2)
    {
        if (argv[1][0] == 'i')
        {
            printf("Enter the test number or 0 to run all tests:\n");
            scanf("%d", &id);
```

```
        }
        else
            id = atoi(argv[1]);
    }
    else
        scanf("%d", &id);

    if (id == 0)
        for (int i = 0; i < SIZE; ++i)
            f[i]();
    else if (0 < id && id <= SIZE)
        f[id - 1]();
    else
        printf("Test %d not found.\n", id);

    return 0;
}

void test1()
{
    int actual = f1(0,1,2,3,4,5);
    int expected = 4;

    if (actual == expected)
        printf("Test 1 : Pass\n");
    else
        printf("Test 1 : Failed (%d)\n", actual);
}

void test2()
{
    int actual = f1(0,1,2,3,44,5);
    int expected = 44;

    if (actual == expected)
        printf("Test 2 : Pass\n");
    else
        printf("Test 2 : Failed (%d)\n", actual);
}

void test3()
{
    int actual = f2(0, 0, 0);
    int expected = 0;

    if (actual == expected)
        printf("Test 3 : Pass\n");
    else
        printf("Test 3 : Failed (%d)\n", actual);
}

void test4()
{
    int actual = f2(10, 20, 30);
    int expected = 60;

    if (actual == expected)
        printf("Test 4 : Pass\n");
```

```
    else
        printf("Test 4 : Failed (%d)\n", actual);
}

void test5()
{
    int actual = f3(0, 1, 2, 3);
    int expected = 1;

    if (actual == expected)
        printf("Test 5 : Pass\n");
    else
        printf("Test 5 : Failed (%d)\n", actual);
}

void test6()
{
    int actual = f3(10, 20, 100, 10);
    int expected = 211;

    if (actual == expected)
        printf("Test 6 : Pass\n");
    else
        printf("Test 6 : Failed (%d)\n", actual);
}

void test7()
{
    int actual = f4(0, 0, 0, 0, 0, 0);
    int expected = 0; /* 0*100000 + 0*10000 + 0*1000 + 0*100 + 0*10 + 0*1 */

    if (actual == expected)
        printf("Test 7 : Pass\n");
    else
        printf("Test 7 : Failed (%d)\n", actual);
}

void test8()
{
    int actual = f4(1, 2, 3, 4, 5, 6);
    int expected = 123456; /* 1*100000 + 2*10000 + 3*1000 + 4*100 + 5*10 + 6*1
*/

    if (actual == expected)
        printf("Test 8 : Pass\n");
    else
        printf("Test 8 : Failed (%d)\n", actual);
}

void test9()
{
    int actual = f5(100000, 0, 0, 0, 0);
    int expected = 1; /* = 100000/100000 - 0/10000 - 0/1000 - 0/100 - 0/10 */

    if (actual == expected)
        printf("Test 9 : Pass\n");
    else
        printf("Test 9 : Failed (%d)\n", actual);
```

```
}

void test10()
{
    int actual = f5(2000000, 20000, 3000, 400, 50);
    int expected = 6;  /* = 2000000/100000 - 20000/10000 - 3000/1000 - 400/100 -
50/10 */

    if (actual == expected)
        printf("Test 10 : Pass\n");
    else
        printf("Test 10 : Failed (%d)\n", actual);
}
```

## Compiling, executing, and testing

Run `make` with the default rule to bring program executable `main` up to date:

```
$ make
```

Or, directly test your implementation by running `make` with target `test`:

```
$ make test
```

If the `diff` command in the `test` rule is not silent, then one or more of your function definitions is incorrect and will require further work.

## File-level documentation

Every **edited by student** source file *must* begin with a *file-level* documentation block. This documentation serves the purpose of providing a reader the purpose of this source file at some later point of time. It has simple format. This module will not use any documentation generator like `Doxygen`.

```
# ------------------------------------------------------------------
# File: functions.asm
# Project: CSD1100 Assignment 10
# Author: Vadim Surov, vsurov@digipen.edu
# Co-Author: Your name, email
# .
# Compile: nasm -f elf64 functions.asm -o functions.o
# Link: gcc main.o functions.o -o main.o -lm
# .
# Copyright: 2021, Digipen Institute of Technology, Singapore
# .
# Note: All functions use at most 6 parameters
#      p1, p2, p3, p4, p5, p6
#      located in registers
#      rdi, rsi, rdx, rcx, r8, r9
#      accordingly.
# ------------------------------------------------------------------
```

## Submission and automatic evaluation

1. In the course web page, click on the appropriate submission page to submit `functions.asm`.
2. Please read the following rubrics to maximize your grade. Your submission will receive:
    - `F` grade if your `functions.asm` doesn't compile with the given options.
    - `F` grade if your `functions.asm` doesn't link to create an executable.
    - Your implementation's output doesn't match correct output of the grader (you can see the inputs and outputs of the auto grader's tests). The auto grader will provide a proportional grade based on how many incorrect results were generated by your submission. `A+` grade if output of function matches correct output of auto grader.
    - A deduction of one letter grade for each incorrect, incomplete or missing documentation block in `functions.asm`. For example, if the automatic grader gave your submission an `A+` grade and one documentation block is missing, your grade will be later reduced from `A+` to `B+`.