Table of Contents

1. Programming in Assembler Laboratory 2
1.1. Laboratory 4 2
1.2. The aim of the exercise 2
1.3. First example code 2
1.4. Second example code 4
1.5. Tasks to be done during laboratory 5
1.6. For the report 5

1. Programming in Assembler Laboratory

1.1. Laboratory 4

Use of vector instructions.

1.2. The aim of the exercise

The aim of the exercise is to learn about vector instructions called SIMD (Single Instruction Multiple Data) and passing parameters to functions as references to common memory. As in previous laboratory exercises, we'll write the dll library with functions called from the C++ program. Functions will calculate the sum, difference, multiplication, and division of four single precision floating point values.



The configuration and solution creation process is identical to the one presented in exercise 2. The main program should be written in C++ and dynamically linked library in assembler.

1.3. First example code

In the first example, we'll write the function in assembler with an internal definition of variables to show how to define floating point vectors in the assembly program. The following code is divided into three parts. The C++ part calls the function in a dynamically linked library. The function, called Sum_of_vectors(), written in the assembler file, calculates the sum of two vectors of four single precision floating point data. They are defined as first_vector and second_vector. The resulting value is stored in the resulting_vector variable. The third file is a def file which exports the function from the assembler source to the C++ program.

C++ file

```
#include <iostream>
#include <windows.h>

typedef int(_stdcall* MyProc)();
HINSTANCE dllHandle = NULL;

int main()
{
    std::cout << "Hello World!\n";
    dllHandle = LoadLibrary(TEXT("JaAsm.dll"));
    MyProc Sum_of_vectors = (MyProc)GetProcAddress(dllHandle,
"Sum_of_vectors");
```

```
    Sum_of_vectors();
}
```

Assembler dll file.

```
;*****************************************************************************
*
;   Vectors definition
;*****************************************************************************
*
.data
first_vector REAL4 2.2, 4.4, 6.6, 8.8
second_vector REAL4 1.1, 1.1, 1.1, 1.1
resulting_vector REAL4 0.0, 0.0, 0.0, 0.0

.code
;*****************************************************************************
*
;   Function to calculate the sum of two vectors
;   No input parameters, no result returned
;*****************************************************************************
*

Sum_of_vectors PROC
    PUSH RSI                            ; RSI is non-volatile register
    LEA RSI, first_vector               ; this helps to find the address
in the memory for dump
    MOVUPS XMM0, OWORD PTR first_vector   ; load address of variable
'SourceDataString' into reg. RSI
    MOVUPS XMM1, OWORD PTR second_vector  ; load address of variable
'DestinBaseString' into reg. RDI
    ADDPS  XMM0, XMM1
    MOVUPS OWORD PTR resulting_vector, XMM0

    POP RSI
    RET                       ; return form procedure
Sum_of_vectors ENDP               ; end of CopyString_base

END
;*****************************************************************************
*
    End of the file
;*****************************************************************************
*
```

Def file.

```
LIBRARY JaAsm
EXPORTS Sum_of_vectors
```

1.4. Second example code

In the second example, we'll write the function in assembler which accepts arguments as three addresses (pointers) to tables of floats defined in a C++ program. The first and second vectors are source while the third argument is a placeholder for the result of calculations. The following code is divided into three parts. The C++ part defines the vectors and calls the function in a dynamically linked library passing the addresses of variables as arguments. The function, called Sum_of_vectors(), written in the assembler file, calculates the sum of two vectors and modifies the content of memory pointed by the third parameter, to return the result. The third file is a def file which exports the function from the assembler source to the C++ program.

C++ file

```
#include <iostream>
#include <windows.h>

typedef float FloatTable[4];
typedef FloatTable* FloatPtr;

typedef int(_stdcall* MyProc)(FloatPtr, FloatPtr, FloatPtr);
HINSTANCE dllHandle = NULL;

FloatTable first_vector { 2.2f, 4.4f, 6.6f, 8.8f };
FloatTable second_vector{ 1.1f, 1.1f, 1.1f, 1.1f };
FloatTable result_vector{ 0.0, 0.0, 0.0, 0.0 };


int main()
{
    std::cout << "Hello World!\n";
    dllHandle = LoadLibrary(TEXT("JaAsm.dll"));
    MyProc Sum_of_vectors = (MyProc)GetProcAddress(dllHandle,
"Sum_of_vectors");
    Sum_of_vectors(&first_vector, &second_vector, &result_vector);
    std::cout << "Result_1 " << (result_vector[0]) << std::endl;
    std::cout << "Result_2 " << (result_vector[1]) << std::endl;
    std::cout << "Result_3 " << (result_vector[2]) << std::endl;
    std::cout << "Result_4 " << (result_vector[3]) << std::endl;
}
```

Assembler file

```
.code

;*****************************************************************************
*
;   Function to calculate the sum of two vectors
;   Accepts two parameters, returns result by third parameter
;*****************************************************************************
*
```

```
Sum_of_vectors PROC
    MOVUPS XMM0, OWORD PTR [RCX]   ; load content of memory pointed by the
first parameter
    MOVUPS XMM1, OWORD PTR [RDX]   ; load content of memory pointed by the
second parameter
    ADDPS  XMM0, XMM1
    MOVUPS OWORD PTR [R8], XMM0    ; store the result into memory pointed by
third parameter

    RET                ; return form procedure
Sum_of_vectors ENDP        ; end of Sum_of_vectors

;*************************************************************************
*

END
```

Def file is the same as in a previous example.

1.5. Tasks to be done during laboratory

1. Compile the examples and observe their behaviour. Set the breakpoint at the beginning of the code and see how the contents of the registers change. Observe XMM registers as calculations are performed there.
2. Write additional functions to calculate subtraction, multiplication, and division of arguments.
3. Write the program which calculates the average of the values in the table using vector (SIMD) instructions. Program assumptions:

- Table contains 64 float values and is defined in C++ file.
- And address of the table is passed to the function as the first argument.
- A size of the table is passed as a second argument (you can assume that the size of the table is a multiply of 4).
- Function uses SIMD instructions.
- Function returns the value as the scalar, traditionally by XMM0.

1.6. For the report

1. Include the code of additional functions.
2. Include the code of the average program (both C++ and asm files).
3. The above programs must contain comments.

From:
https://dokuwiki.zmitac.aei.polsl.pl/ - **RAU6 Dokuwiki**

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s1:apl:lab:lab4

Last update: **2024/12/05 12:39**