Table of Contents

1. Programming in Assembler Laboratory 2
1.1. Laboratory 3 2
1.2. The aim of the exercise 2
1.3. The example code 2
1.4. Tasks to be done during laboratory 4

1. Programming in Assembler Laboratory

1.1. Laboratory 3

Addressing modes.

1.2. The aim of the exercise

The aim of the exercise is to learn about different addressing modes you can use to access the data in the memory.

1.3. The example code

The following code is divided into two parts. The C++ part calls the functions in a dynamically linked library. DLL contains three different functions searching for the character using different addressing modes. One function, called FindChar_1() searches for the “J” character in a string using an index addressing mode. The second function searches for the character using base + index addressing mode. The last one uses base + index + displacement.

The string is defined inside the DLL.

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"));
    std::cout << procedura();

    MyProc FindChar = (MyProc)GetProcAddress(dllHandle, "FindChar_1");
    MyProc ReadTime = (MyProc)GetProcAddress(dllHandle, "ReadTime_1");

    char szString[] = { 'A','G','I','K','S','Z','J', 0xFF }; // string
definition
    printf("FindChar_1 %s\n", (FindChar() == 1 ? "Found J" : "Not Found
J"));
    printf("Processing time: %i\n", ReadTime());
```

```
}
```

Assembler dll file.

```
;************************************************************************
*
;   String definition
;************************************************************************
*
.data
DataString DB 'AGIKSZJ', 0FFH; string definition

.code
;************************************************************************
*
;   Procedure to find the character in a string
;   No input parameters, result is 1 if character found, 0 otherwise
;************************************************************************
*
FindChar_1 PROC
    PUSH RSI
    LEA RSI, DataString      ; load the address of variable 'DataString'
into reg. ESI
    MOV AL, 'J'              ; load the code of character 'J' into reg. AH
Check_End:
    CMP BYTE PTR [RSI], 0FFH ; is it the end of a string (special character
FF)?
    JE Not_Find              ; found end character (guard)
    CMP AL, [RSI]            ; compare a character with chosen char of
string 'DataString'
    JE Got_Equal             ; character found!
    ADD RSI, 1               ; increment an offset
    JMP Check_End            ; search loop
Got_Equal:
    MOV DL, [RSI]            ; load found char into DL
    JMP Done
Not_Find:
    MOV RAX,0                ; character not found
    POP RSI
    RET                      ; return from procedure
Done:
    MOV RAX,1                ; character found
    POP RSI
    RET                      ; return form procedure
FindChar_1 ENDP              ; end of FindChar_1

;************************************************************************
*
;   Procedure to measure the time of other function execution
;   No input parameters, result is the time (number of processor ticks)
;************************************************************************
```

```
*
ReadTime_1 PROC
    RDTSC                    ; read of time stamp counter
    MOV RCX,RAX

    CALL FindChar_1          ; call to the tested function

    RDTSC
    SUB RAX,RCX

    RET                      ; return form procedure
ReadTime_1 ENDP

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

1.4. Tasks to be done during laboratory

1. Compile the example code and observe its behaviour. Set the breakpoint at the beginning of the code and see how the contents of the registers change with every loop pass.
2. Make the function more universal: define the string in a C++ file and pass its address as the parameter to the function. Pass the character as the second parameter.
3. Rewrite the search algorithm with vector instructions (SSE, or AVX). Suggest what to do if the string length is not a multiple of register size.
4. Compare execution time for scalar (normal) instructions and vector instruction for the same string.

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

Permanent link:
https://dokuwiki.zmitac.aei.polsl.pl/en:edu:zmitac:s1:apl:lab:lab3?rev=1732887812

Last update: **2024/11/29 14:43**