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 5
1.5. For the report 5

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 copying the content of the source string to three destination strings using different addressing modes. One function, called CopyString_base() uses a base addressing mode. The second function CopyString_base_index() uses base + index addressing mode. The last one CopyString_base_disp() uses base + displacement.

The strings are 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"));

    MyProc Copy_base = (MyProc)GetProcAddress(dllHandle, "CopyString_base");
    MyProc Copy_index = (MyProc)GetProcAddress(dllHandle,
"CopyString_base_index");
    MyProc Copy_disp = (MyProc)GetProcAddress(dllHandle,
"CopyString_base_disp");

    Copy_base();
    Copy_index();
    Copy_disp();
```

```
}
```

Assembler dll file.

```
.data
SourceDataString DB 'AGIKSZJ', 0FFH   ; source string definition
DestinBaseString DB '       ', 0; buffer for destination string, null
terminated
DestinIndexString DB '       ', 0; buffer for destination string, null
terminated
DestinDisplString DB '       ', 0; buffer for destination string, null
terminated

.code

;*************************************************************************
*
;   This function uses base addressing mode
;*************************************************************************
*
CopyString_base PROC
    PUSH RSI                    ; RSI, RDI are non-volatile registers
    PUSH RDI
    LEA RSI, SourceDataString   ; load address of variable
'SourceDataString' into reg. RSI
    LEA RDI, DestinBaseString   ; load address of variable
'DestinBaseString' into reg. RDI
    MOV AL, 0FFH                ; load the ending character 'FF' into reg.
AL
Copy_Loop:
    CMP BYTE PTR [RSI], AL      ; is it the end of a string (special
character FF)?
    JE String_End               ; found end character (guard)
    MOV AH, [RSI]               ; take one character from source string
    MOV [RDI], AH               ; put it into  destination string
    ADD RSI, 1                  ; increment index registers
    ADD RDI, 1
    JMP Copy_Loop               ; jump to beginning of a loop
String_End:
    POP RDI
    POP RSI
    RET                         ; return form procedure
CopyString_base ENDP            ; end of CopyString_base

;*************************************************************************
*
;   This function uses base + index addressing mode
;*************************************************************************
*
CopyString_base_index PROC
    PUSH RSI                    ; RSI, RDI are non-volatile registers
```

```
    PUSH RDI
    LEA RSI, SourceDataString   ; load address of variable
'SourceDataString' into base reg. RSI
    LEA RDI, DestinIndexString  ; load address of variable
'DestinIndexString' into base reg. RDI
    MOV AL, 0FFH                ; load the ending character 'FF' into reg.
AL
    XOR RCX, RCX                ; zero index register RCX, it is volatile
Copy_Loop:
    CMP BYTE PTR [RSI+RCX], AL  ; is it the end of a string (special
character FF)?
    JE String_End               ; found end character (guard)
    MOV AH, [RSI+RCX]           ; take one character from source string
    MOV [RDI+RCX], AH           ; put it into  destination string
    ADD RCX, 1                  ; increment index register
    JMP Copy_Loop               ; jump to beginning of a loop
String_End:
    POP RDI
    POP RSI
    RET                         ; return form procedure
CopyString_base_index ENDP      ; end of CopyString_base_index

;*************************************************************************
*
;   This function uses base addressing mode with displacement
;*************************************************************************
*
CopyString_base_disp PROC

; Calcucate the displacement of destination string from source
    Address_difference EQU DestinDisplString - SourceDataString

    PUSH RSI                    ; RSI, RDI are non-volatile registers
    LEA RSI, SourceDataString   ; load address of variable
'SourceDataString' into base reg. RSI
    MOV AL, 0FFH                ; load the ending character 'FF' into reg.
AL
Copy_Loop:
    CMP BYTE PTR [RSI], AL      ; is it the end of a string (special
character FF)?
    JE String_End               ; found end character (guard)
    MOV AH, [RSI]               ; take one character from source string
    MOV [RSI+Address_difference], AH   ; put it into destination string
    ADD RSI, 1                  ; increment index register
    JMP Copy_Loop               ; jump to beginning of a loop
String_End:
    POP RSI
    RET                         ; return form procedure
CopyString_base_disp ENDP       ; end of CopyString_base_disp

;*************************************************************************
```

```
*

END
```

def file

```
LIBRARY JaAsm
EXPORTS CopyString_base
EXPORTS CopyString_base_index
EXPORTS CopyString_base_disp
```

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. Observe the content of the memory at the beginning address of the source string. Look if other strings are filled properly.
3. Write the version of the assembler function which copies the source to a destination string using chosen string instruction with repetition (use REP prefix).

1.5. For the report

1. Put in the report a screenshot of the window with the results displayed (memory dump with copied texts).
2. Explain differences in addressing modes used in the functions. Explain what is the result of the line:

```
Address_difference EQU DestinDisplString - SourceDataString
```

3. Include the code of your new function written in assembler WITH COMMENTS. Also, add a screenshot of the window with the results displayed (copied texts).

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=1733233215

Last update: **2024/12/03 14:40**