MINISTERUL EDUCAȚIEI, CULTURII ȘI CERCETĂRII AL REPUBLICII MOLDOVA Universitatea Tehnică a Moldovei Facultatea Calculatoare, Informatică şi Microelectronică Departamentul Inginerie Software și Automatică

Copta Adrian | FAF-223

Report

Laboratory work n.5

Computer Architectures

Verificat:

Voitcovschi Vladislav *asist.univ*

Chișinău – 2024

1. Purpose of the task work:

For this lab, each student must create an NASM assembler program that contains 10 cyclic processes (functions). Additionally, the program must allow the user to choose which of the 10 processes to execute at program launch.

To accomplish this task, follow the steps below:

• Write an interactive menu that allows the user to choose from the 10 processes.

• Write the code for each of the 10 processes. Each process must be written cyclically so that the program always returns to the interactive menu after a process is completed.

• Ensure that your program is well-commented and structured clearly so that it is easy to understand and modify

• Test the program to ensure that it works correctly and that the user can choose any of the 10 processes • Ensure that each student personalizes their program in a unique way so that there are no identical programs.

• Prior to presenting the lab, each student must present their program and demonstrate that it can be used to choose any of the 10 processes.

❖ The first program will contain a generator of 10 random numbers from 1 to 55

❖ These will be the varinates of each

2. Introduction:

In the realm of computer science and software engineering, understanding the fundamental concepts of assembly language programming is paramount. Assembly language serves as a bridge between high-level programming languages and the machine code executed by the central processing unit (CPU). In this laboratory work report, we delve into the basics of assembly language and its implementation using NASM (Netwide Assembler).

Assembly language, often referred to simply as assembly, provides a low-level representation of computer programs. Unlike high-level languages such as Python or C, assembly language directly corresponds to the machine instructions understood by the CPU. It offers programmers precise control over hardware resources and facilitates optimization for performance-critical applications.

NASM, a widely used assembler in the realm of x86 and x86-64 architectures, plays a pivotal role in translating assembly code into machine code. It provides a robust set of features and directives that aid in the development of efficient and portable assembly programs. In this report, we will

explore the fundamental concepts of assembly language programming, elucidate the role of NASM in assembling code, and demonstrate practical examples to reinforce comprehension.

By the end of this laboratory work, readers will have gained a solid foundation in assembly programming and proficiency in utilizing NASM as a tool for code development.

3. Implementation:

Lab5 program:

Here is the beginning of the program in assembler language.

```
        .section .rdata,"dr"
        .align 4
LC0:
        .ascii "0. Generate 10 random numbers between 1 and 55\0"
LC1:
        .ascii "1. Add two numbers\0"
LC2:
        .ascii "2. Subtract two numbers\0"
LC3:
        .ascii "3. Multiply two numbers\0"
```

```
LC4:
	.ascii "4. Divide two numbers\0"
LC5:
	.ascii "5. Generate random number\0"
LC6:
	.ascii "6. Decimal to binary\0"
LC7:
	.ascii "7. Decimal to hexadecimal\0"
LC8:
	.ascii "8. Concatenate two strings\0"
LC9:
	.ascii "9. Compare two strings\0"
LC10:
	.ascii "q. Exit the program\0"
	.text
	.p2align 4,,15
	.globl	_help
	.def	_help;	.scl	2;	.type	32;	.endef
_help:
	subl	$28, %esp
	movl	$LC0, (%esp)
	call	_puts
	movl	$LC1, (%esp)
```

```
	call	_puts
	movl	$LC2, (%esp)
	call	_puts
	movl	$LC3, (%esp)
	call	_puts
	movl	$LC4, (%esp)
	call	_puts
	movl	$LC5, (%esp)
	call	_puts
	movl	$LC6, (%esp)
	call	_puts
	movl	$LC7, (%esp)
	call	_puts
	movl	$LC8, (%esp)
	call	_puts
	movl	$LC9, (%esp)
	call	_puts
	movl	$LC10, (%esp)
	call	_puts
	addl	$28, %esp
	ret
	.section .rdata,"dr"
LC11:
```

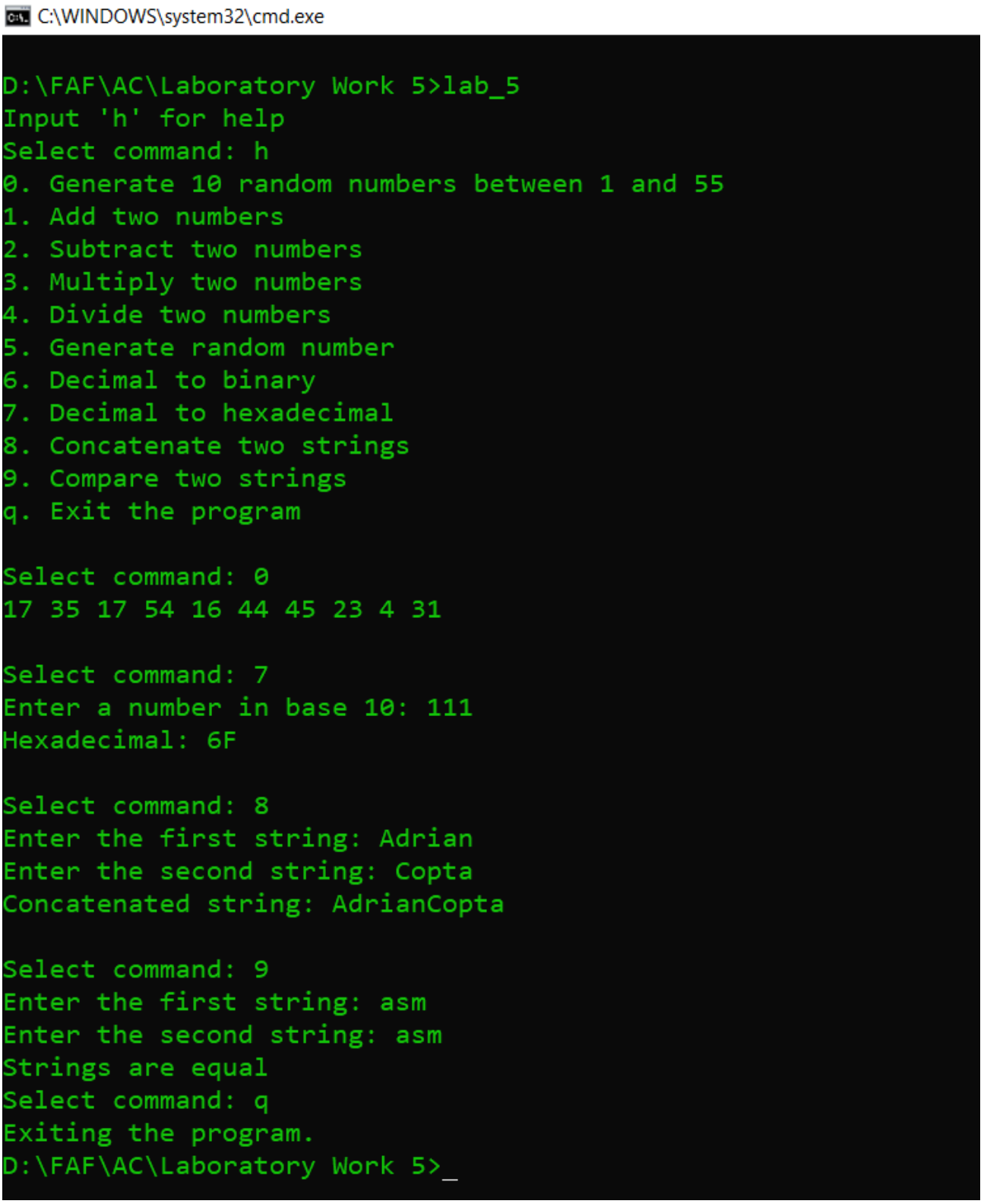
```
	.ascii "%d \0"
	.text
	.p2align 4,,15
	.globl	_generate_rand
	.def	_generate_rand; .scl	2;	.type	32;	.endef
_generate_rand:
	pushl	%edi
	pushl	%esi
	xorl	%edi, %edi
	pushl	%ebx
	movl	$-1840700269, %esi
	subl	$16, %esp
	movl	32(%esp), %ebx
	movl	$0, (%esp)
	call	_time
	movl	%eax, (%esp)
	call	_srand
	testl	%ebx, %ebx
	jle	L6
	.p2align 4,,10
L7:
	call	_rand
	movl	%eax, %ecx
```

```
        movl  $LC11, (%esp)
        addl  $1, %edi
        imull %esi
        movl  %ecx, %eax
        sarl  $31, %eax
        addl  %ecx, %edx
        sarl  $5, %edx
        subl  %eax, %edx
        imull $56, %edx, %edx
        subl  %edx, %ecx
        movl  %ecx, 4(%esp)
        call  _printf
        cmpl  %edi, %ebx
        jne   L7
L6:
        movl  $10, 32(%esp)
        addl  $16, %esp
        popl  %ebx
        popl  %esi
        popl  %edi
        jmp   _putchar
        .section .rdata,"dr"
LC12:
```

```
	.ascii "Enter the first number: \0"
LC13:
	.ascii "%d\0"
LC14:
	.ascii "Enter the second number: \0"
LC15:
	.ascii "Sum: %d\12\0"
	.text
	.p2align 4,,15
	.globl	_add_two_numbers
	.def	_add_two_numbers;	.scl	2;	.type	32;	.endef
_add_two_numbers:
	subl	$44, %esp
	movl	$LC12, (%esp)
	call	_printf
	leal	24(%esp), %eax
	movl	$LC13, (%esp)
	movl	%eax, 4(%esp)
	call	_scanf
	movl	$LC14, (%esp)
	call	_printf
	leal	28(%esp), %eax
	movl	$LC13, (%esp)
```

```
	movl	%eax, 4(%esp)
	call	_scanf
	movl	28(%esp), %eax
	addl	24(%esp), %eax
	movl	$LC15, (%esp)
	movl	%eax, 4(%esp)
	call	_printf
	addl	$44, %esp
	ret
	.section .rdata,"dr"
```

Output:



4. **Conclusion**

The assembly code provided offers a comprehensive implementation of a command-line interface program with various functionalities related to mathematical operations and number conversions. It demonstrates the usage of conditional jumps, function calls, and data manipulation to create an interactive user experience. The program's modular structure, with distinct command functions and helper functions, enhances readability and maintainability. Additionally, the inclusion of error handling and a help message contributes to user-friendliness. Overall, the code showcases a well-structured and functional approach to implementing a command-line utility in assembly language, suitable for educational purposes or small-scale practical applications.