|  |  |
| --- | --- |
| Student name and surname |  |

EFREI CA2023F

Computer Architecture – Assembly language programming part 2

In this lab exercise we will continue on assembly language programming for x86 processors working in 32-bit mode. To do this exercise we are going to use SASM environment: (<https://dman95.github.io/SASM/english.html>).

Useful information about AT&T syntax and x86-32 assembly language instructions can be found at <https://gist.github.com/mishurov/6bcf04df329973c15044> .

# Subprograms

A very useful feature of high level programming languages is possibility to create reusable code (that parts of code we call functions or procedures). Assembly language also offers this possibility. There are two processor instructions dedicated for this: **call** and **ret**. Here is an example how to use these instructions:

##########################################################################

#

# Program: call

#

# Date: 08/03/2023

#

# Author: WZ

#

# Purpose: A simple subprogram demo in x86 assembly for GAS

#

##########################################################################

.extern \_printf

.global \_main # the entry point to the program

# must be specified for linker.

.data # .data section declaration

msg:

.asciz "Hello from subprogram. \n" # Declare a label "msg" which has

# string we want to print.

.text # .text section declaration

\_main:

movl %esp, %ebp # for correct debugging

call print\_msg # call the subprogram that prints message

call print\_msg # call the subprogram that prints message

# end program

movl $0x00, %eax

ret

# subprogram print\_msg that prints msg to STDOUT

# subprograms must be placed before entry point of the program (\_main:) or

# after instruction that ends the program and return to the OS

print\_msg: # label that identifies the subprogram

pushl $msg # put pfintf() function parameter

# (pointer to the msg string) on the stack

call \_printf # invoking the printf() function

addl $4, %esp # before calling printf we put 1 longword (4 bytes)

# on the stack, now we must take them out

ret # end of the subprogram - return to the next instruction

# after call

# it must be the last instruction of any subprogram.

When you are writing your own subprograms there is a good practice: at the begin of the subprogram you should protect the contents of all registers used by subprogram storing them on the stack and restore their values just before the ret instruction.

## Assignment 1

Analyze the subprogram example and write own program that prints two different strings. **Each string should be printed by its own subprogram.**

Upload source code of a properly working program on the moodle platform as a solution of this assignment. Please remember to place your name in commentary at the beginning of the source code (Author section).

# Simple loop

The program below demonstrates how to create and use simple counter loop in x86 assembly.

##########################################################################

#

# Program: loop

#

# Date: 08/03/2023

#

# Author: WZ

#

# Purpose: A simple counter loop demo program in x86 assembly for GAS

#

##########################################################################

.data # .data section declaration

msg:

.asciz "Simple loop demo. \n" # Declare a label "msg" which has

# string we want to print.

.text # .text section declaration

.global \_main # entry point

\_main:

movl $10,%ecx # %ecx works as loop counter.

# we put here how many times loop should repeat

l1: # label l1 points to the first instruction of the loop

push %ecx # save loop counter on the stack

# in case if we need to use %ecx register

# for something else

# printing message

pushl $msg # put pfintf() function parameter

# (pointer to the msg string) on the stack

call \_printf # invoking the printf() function

addl $4, %esp # before calling printf we put 1 longword (4 bytes)

# on the stack, now we must take them out

pop %ecx # restore counter value just before it’s needed

loop l1 # close the loop.

# decrease loop counter (%ecx)

# and if %ecx <> 0 go to label l1 to run loop again

# end program

movl $0x00, %eax

ret

As you can see, the x86 processor has a special instruction (loop) used to create counter loops . Important element of execution of this instruction is %ecx register, which holds the loop counter. It can cause problems when creating nested loops.

## Assignment 3 – Loop assignment

Having declared two strings:

.data

str1: .asciz "0000\n"

str2: .asciz " 1111\n"

write a program which displays the following pattern on the console:

0000

0000

0000

1111

0000

0000

0000

1111

0000

0000

0000

1111

Please remember that the **program must use subprograms for displaying each string** and at least one for loop (minimum score for this task).

**The most valuable solution should use two nested for loops (maximum score).**

Upload source code of a properly working program on the moodle platform as a solution of this assignment. Please remember to place your name in commentary at the beginning of the source code (Author section).