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

EFREI CA2023S

Computer Architecture – Assembly language programming

In this lab exercise we focus on assembly language programming for x86 processors working in 32-bit mode.

What is the assembly language?

It’s a low level programming language that’s allow the programmer to operate directly at processor instructions. One assembler command correspond to exactly one processor instruction. This language is usually different for different processors.

During this course **GNU Assembler** (gas) will be used as a compiler and programs will be written with **AT&T** x86-32 (IA-32) assembly language syntax. Useful information about that syntax and x86-32 assembly language instructions can be found at <https://gist.github.com/mishurov/6bcf04df329973c15044> .

To do this exercise we are going to use SASM environment: (<https://dman95.github.io/SASM/english.html>).

It contains all tools (editor, compiler linker and debugger) required to write and run own assembly programs.

# Configuring the SAMS environment

Before you start doing this exercise, some configuration of SASM environment is required. After starting the SASM environment go to the Settings menu and find the Build tab. On that tab check if x86 is checked in Mode section and GAS is checked in Assembler section. Correct settings are shown on figure below.

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, wyświetlacz

Opis wygenerowany automatycznie

# First assembly language program – Hello\_world

In SASM editor create new file and paste following code (you can also find this code in **hello.asm** file):

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

#

# Program: hello

#

# Date: 08/03/2023

#

# Author: WZ

#

# Purpose: A simple hello world program in x86 assembly for GAS

#

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

.data # .data section declaration

msg:

.asciz "Hello, world!\n" # Declare a label "msg" which has

# string we want to print.

.extern \_printf

.text # .text section declaration

.global \_main # entry point

\_main: # label that identifies entry point

movl %esp, %ebp # for correct debugging contetnt of %esp reg

# is placed into %ebp

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

# end program

movl $0x00, %eax

ret

When the code is ready , it may be executed by selecting „Build and run” command from SASM Build menu. If everythink is correct ,a „Hello world” message should apear in SASM outuput window.

# Assignment 1 - Displaying own message

Based on the code from , write program that display on the console following message:

**###################################**

**# GNU ASM EFREI AGH FALL 2023 #**

**###################################**

Submit the 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).

# Using standard program I/O functions from C library in assembly programs

In programs written in assembly language, communication between the program and the user usually is carried out through functions provided by the operating system or using standard libraries available in higher-level languages (like C).We can easy print a different types of data using standard **printf()** function and read different types of data using standard **scanf()** function.

## Printing user defined numbers

The source code of example program that prints string and user defined integer value on console using standard C printf() function is available in **printax.asm** file.

In **.data** section there are two null-terminated ASCII strings are declared:

**msg: .asciz "Now we can easy print value of EAX register. \n"**

**msg2: .asciz "EAX value is %d \n"**

They will be printed o console using printf() function from C language library.

To invoke printf()function the ***call*** instruction is used, but before it can be done, the required parameters must be delivered. In the calling convention used in C, the stack is used to deliver the arguments, so before the **call printf** instruction, the **push** instruction or instructions are used to place the argument on the stack:

**pushl $msg # put pfintf() parameter (pointer to the msg string)**

**# on the stack**

If we want to print simple null-terminated ASCII string , the pointer to that string must be placed at the top of the stack, and this in only one required argument for this operation. So after that , the **call printf** can be made:

**call \_printf # execute printf()**

After return from the print() function we need to remove our useless now parameters from the stack (it’s enough to increase by 4 value of the %esp register):

**addl $4,%esp # we put 1 long word (4 bytes) on the stack,**

**# now we must take them out**

Second call of printf() function demonstrates how to print a value of integer number stored in one of the processor’s register.

In this case it’s necessary to deliver 2 arguments:

* format string that tell what to print (msg2 string will be used) and
* a value that should be printed (a value stored in EAX register will be used).

First, a value 0x55AA is placed into %eax register

**movl $0x55AA,%eax # put a hex value 0x55AA into %eax**

Next, the arguments for printf() function are prepared. It should be remembered that parameters for printf() should be placed on the stack in the right order (last parameter is at the top of the stack), so they are pushed in reverse order:

pushl %eax # Parameter 2 - value that we want to print

pushl $msg2 # Parameter 1 - format string that tell what print

It should be remembered that after the execution of printf() call, the top of the stack must be cleaned before the program can continue. Because this time, two long word (8 bytes) arguments were pushed onto the stack, to clean it up %esp register must be increased by 8:

**addl $8,%esp # we put 2 long word (8 bytes) on the stack,**

**# now we must take them out**

At the end, the Exit program instruction sequence is invoked to finish the program.

Compile and execute the program. At the output window you should see text similar to this:

You should see

## Assignment 2 – using printf () function

Please recall what formatting string is used when calling the printf() function in C to make the integer number displayed in hexadecimal instead of decimal.

Modify example program **printax.asm** to get content of EAX register in both decimal and hexadecimal format.

Output of the program should be similar to this:

**EAX value is 21931 in decimal which is 55AB in HEX.**

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).

## Reading integer number from user

The source code of example program that reads integer number from user and prints them on the console is available in **readint.asm** file. In this example a scanf() function is used to read integer number from console.

Compile and execute that program in SASM.

Investigate the source code to learn how to use scanf() function.

## Assignment 3 - reading float number from user

Using program **readint.asm** as an example, write own program that read from user single precision float number then displays at the console its representation in hexadecimal.

Example output of the program should look like this:

**Please enter any FP number: 12.3**

**The number is represented by 4144cccd.**

Size of single precision float number in x86 is 32 bits. In can be stored in .float value and its representation perfectly fits in any universal register e.g. %eax.

You may use “**%X**” format string to easy get the hexadecimal representation of the value.

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).

## Assignment 4 - Floating point number representation

Execute again program created in 4.4 and this time enter a different than previously non-integer number.

Write down entered number and it’s hexadecimal representation:

|  |  |
| --- | --- |
| Your FP number: | hexadecimal representation: |
|  |  |

Please find in the lectures (or in different sources) information about floating point number representation in x86 processors then prepare a short note that explain in details why received hexadecimal number is a representation of the entered floating point number. Upload the note on the moodle platform as a solution of this assignment. Please remember to place your name in commentary at the beginning of the note.