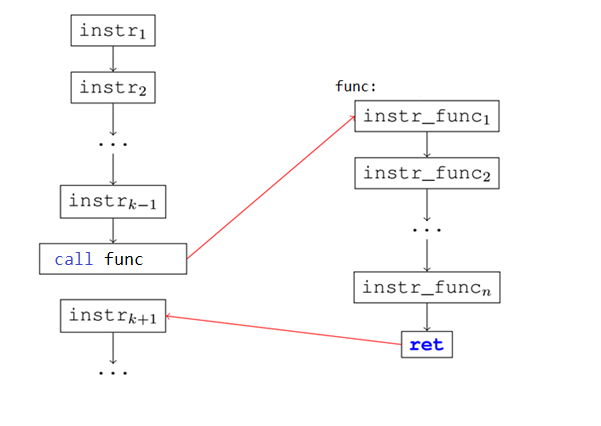
# Seminar 5

**Library functions call**

In order to call functions from a library (e.g. a .dll or .lib library), we need to use the

call [functionname]

instruction which pushes the current memory address (i.e. the Return Address) on the stack and performs a jump to the starting address of functionname. This allows a RET instruction to pop the return address into either CS:IP or IP only (depending on whether it is a Near or Far call) and thus return control to the instruction immediately after the CALL instruction.



Before we call the function we need to pass the actual parameters to the function. The parameters are passed to the function using the stack using the cdecl calling convention (although there are other calling conventions that can be used). This calling convention has the following rules:

* The parameters are passed on the stack from right to left; an element of the stack is a dword
* The default result is returned by the function in EAX
* The EAX, ECX, EDX registers can be modified in the body of the function (there is no warranty that they keep their initial value (i.e. the value they had before entering the function) when exiting the function.
* The function will not free the parameters from the stack; it is the responsibility of the calling code

A list of C run-time library functions (i.e. functions of the msvcrt.dll library) can be found here:

[*https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/crt-alphabetical-function-reference?view=vs-2017*](https://docs.microsoft.com/en-us/cpp/c-runtime-library/reference/crt-alphabetical-function-reference?view=vs-2017)

For printing something on the screen, we will use the function *printf()*. The syntax of this function is:

*printf (string format, value1, value2, … )*

where *format* is a string that specifies what is printed on the screen and *value1, value2 …* are values (bytes, words, dwords, strings). Every character that appears in *format* is printed on the screen exactly as it is, except the characters that are preceeded by ‘%’ which will be replaced by values from the *value1, value2 …* list. The first character preceeded by ‘%’ from *format* will be replaced when printed with *value1*, the second character preceeded by ‘%’ from *format* will be replaced when printed with *value2*, … In assembly, any value from the values list can be a constant or a variable. If it is a constant or a variable different than string, its value will be placed on the stack. If the value is a variable of type string, its offset will be placed on the stack. Below there are some examples:

*printf(“a=%d”, x)* - prints on the screen “a=[value of x]”

*printf(“%d + %d=%d”,a,b,c)* - prints on the screen “[value of a] + [value of b] = [value of c]”

*printf(“%s %d”, s, a)* - prints on the screen “[string s] [value of a]”.

Conversly, we use the function *scanf()* for reading from the keyboard. The syntax is:

*scanf (string format, variable1, variable2, … )*

where *format* is a string that specifies what is read from the keyboard and *variable1, variable2 …* are offsets of variables in assembly (of types bytes, words, dwords, strings). The *format* string should only contain ‘%’ characters followed by a type specification like %d - decimal, %s - string, %c – character. The first ‘%’ expression describes the type of the first value that is read and set to *variable1*. The second ‘%’ expression describes the type of the second value that is read and set to *variable2*. Etc. Some examples below:

*scanf(“%d %d”, a, b*) - reads two integer/decimal values and sets them to a and b

*scanf(“%s”, s*) - reads a string into variable s

Ex.1. The code below will print the message ”n=” on the screen and then will read from the keyboard the value for the number n.

bits 32

global start

extern exit, printf, scanf ; exit, printf and scanf are external functions

import exit msvcrt.dll

import printf msvcrt.dll ; tell the assembler that function printf is in msvcrt.dll

import scanf msvcrt.dll ;

segment data use32 class=data

n dd 0

message db "n=", 0 ; strings for C functions must end with ZERO (ASCIIZ strings)

format db "%d", 0 ; strings for C functions must end with ZERO (ASCIIZ strings)

segment code use32 class=code

start:

; calling printf(message) => "n=" will be printed on the screen

push dword message ; we store the offset of message (not its value) on the stack

call [printf] ; call printf

add esp, 4\*1 ; free parameters from the stack; 4 = dword size in bytes

; 1 = number of parameters

; remember that the stack grows towards small addresses and the elements of the stack are dwords.

; that is, assuming the dword from the top of the stack is at address ADR, by pushing another dword

; on top of the stack, the new dword is on address ADR-4. ESP always points to the top of the stack.

; we clear/free 4 bytes from the top of the stack by “add ESP, 4”

; call scanf(format, n) => read a decimal number in variable n

; parameters are placed on the stack from right to left

push dword n ; push the offset of n

push dword format ; push the offset of format

call [scanf] ;

add esp, 4 \* 2 ; free 2 dwords from the stack

; call exit(0)

push dword 0 ; punem pe stiva parametrul pentru exit

call [exit] ; apelam exit pentru a incheia programul

Ex.2. A program that reads 2 numbers, a and b, computes their sum and prints it on the screen.

bits 32

global start

extern exit, printf, scanf

import exit msvcrt.dll

import printf msvcrt.dll

import scanf msvcrt.dll

segment data use32 class=data

a dd 0

b dd 0

result dd 0

format1 db ‘a=’, 0 ; all formats used for scanf/printf are required to be ASCIIZ strings

format2 db ‘b=’, 0 ; all formats used for scanf/printf are required to be ASCIIZ strings

readformat db ‘%d’, 0 ; all formats used for scanf/printf are required to be ASCIIZ strings

printformat db ‘%d + %d = %d\n’, 0 ; all formats are required to be ASCIIZ strings

segment code use32 class=code

start:

; call printf(“a=”)

push dword format1

call [printf]

add esp, 4\*1

; call scanf(“%d”, a)

push dword a ; push the offset of a for reading (not its value)

push dword readformat

call [scanf]

add esp, 4\*2

; call printf(“b=”)

push dword format2

call [printf]

add esp, 4\*1

; call scanf(“%d”, b)

push dword b ; push the offset of a for reading (not its value)

push dword readformat

call [scanf]

add esp, 4\*2

mov eax, [a]

add eax, [b]

mov [result], eax

; call printf(“%d + %d = %d\n”, a, b, result)

push dword [result] ; push the value of result for printing

push dword [b] ; push the value of b for printing

push dword [a] ; push the value of a for printing

push dword printformat

call [printf]

add esp,4\*4

push dword 0

call [exit]

The C source for the program is:

#include<stdio.h>

int main(){

int n, m, s;

printf("Give the first number:\n"); //13,10,0 NL CR

scanf("%d", &n);

printf("Give the second number:\n");

scanf("%d", &m);

s=n+m;

printf("The sum of %d and %d is %d \n",n, m , s);

return 0;

}

Working with files

|  |
| --- |
| 1. **Open a file**   **FILE \* fopen(const char\* filename, const char \* access\_mode)** |
| ARGUMENTS |
| |  |  |  | | --- | --- | --- | | Mode | Meaning | Description | | r | read | - Open file for reading. - The file must exist. | | w | write | - If the file does not exist, it creates a new file and opens it for writing.  - If a file with the given name exists, it opens it for writing. It overwrites the content of the file. | | a | append | - If the file does not exist, it creates a new file and opens it for writing.  - If a file with the given name exists, it opens it for writing. It does not overwrite the content, it continues writing at the end of the file. | | r+ | Read+write for existing file | - Open file for reading and writing. - The file must exist. | | w+ | Read+write | - If the file does not exist, it creates a new file and opens it for reading and writing.  - If a file with the given name exists, it opens it for reading and writing. It overwrites the content of the file | | a+ | Read+append | - If the file does not exist, it creates a new file and opens it for reading and writing.  - If a file with the given name exists, it opens it for reading and writing. It does not overwrite the content, it continues writing at the end of the file. | |
| RESULTS |
| If the file is successfully opened, **EAX will contain the file descriptor** (an identifier) which can be used for working with the file (reading and writing). **If an error occurs, fopen will set EAX to 0** |

|  |
| --- |
| 1. **Write into a file**   **int fprintf(FILE \* stream, const char \* format, <variable\_1>, < variable \_2>, <...>)** |
| **RESULT** |
| In case of an **error**, **EAX** contains a value **< 0** |

|  |
| --- |
| 1. **Read from a file**   **int fread(void \* str, int size, int count, FILE \* stream)** |
| * First argument is the string where the bytes that are read from the file are stored * Second argument represents the size of the elements that are read from the file * Third argument represents the maximum number of elements to be read * Last argument is the file descriptor |
| **RESULT** |
| EAX will contain the number of elements read. If this number is below **count**, it means wither that there was an error, or that the function got to the end of the file. |

|  |
| --- |
| 1. **Closing an open file**   **int fclose(FILE \* descriptor)** |

Ex. 3

; **This program reads the content of a text file (a.txt), adds 1 to each byte and then writes**

**; these bytes to a new file (b.txt)** and then renames this new file to be the old file name (a.txt).

bits 32

global start

; declare external functions needed by our program

extern exit, perror, fopen, fclose, fread, fwrite, rename, remove

import exit msvcrt.dll

import fopen msvcrt.dll

import fread msvcrt.dll

import fwrite msvcrt.dll

import fclose msvcrt.dll

import rename msvcrt.dll

import remove msvcrt.dll

import perror msvcrt.dll

segment data use32 class=data

inputfile db 'a.txt', 0

outputfile db 'b.txt', 0

modread db 'r', 0

modwrite db 'w', 0

c db 0

handle1 dd -1

handle2 dd -1

eroare db 'error:', 0

segment code use32 class=code

start:

; fopen(string path, string mode) - opens the file *path* in the specified *mode*. *mode* can be “r”

; for reading the file or “w” for writing the file

push dword modread ; for strings, the offset is pushed on the stack

push dword inputfile ; for strings, the offset is pushed on the stack

call [fopen]

add esp, 4\*2

; fopen returns in EAX the file handle or zero (in case of error)

; this file handle is just a dword used by the operating system and is required for all subsequent

; function calls that work with this file.

mov [handle1], eax ; store the handle in a local variable

cmp eax, 0

je theend ; if error, move to the end of the program

; fopen(string path, string mode)

push dword modwrite ; open the outputfile for writting

push dword outputfile

call [fopen]

add esp, 4\*2

; fopen returns in EAX the file handle or zero (in case of error)

mov [handle2], eax ; store the second handle in a local variable

cmp eax, 0

je theend

repeat:

;fread(string ptr, integer size, integer n, FILE \* handle) - reads *n* times *size* bytes from the

; file identified by *handle* and place the read bytes in the string *ptr.*

; we read 1 byte from the file handle1

push dword [handle1] ; read from handle1

push dword 1 ; read 1 time

push dword 1 ; read 1 byte

push dword c ; store the byte in c

call [fread]

add esp, 4\*4

cmp eax, 0 ; the function returns zero in EAX in case of error

je error

add byte [c], 1

;fwrite(string ptr, integer size, integer n, FILE \* handle) - writes *n* times *size* bytes from

; the string *ptr* into the file identified by *handle*.

; write 1 byte in file handle2

push dword [handle2] ; write into file handle 2

push dword 1 ; write 1 time

push dword 1 ; write 1 byte

push dword c ; from c

call [fwrite]

add esp, 4\*4

cmp eax, 0

je error

jmp repeat

error:

; fclose(FILE\* handle) - close the file identified by handle

push dword [handle1]

call [fclose]

add esp, 4\*1

; fclose(FILE\* handle) - close the file identified by handle

push dword [handle2]

call [fclose]

add esp, 4\*1

; remove( string path ) - remove the file *path*

push dword inputfile

call [remove]

add esp, 4\*1

; rename( string oldname, string newname ) - rename the fine *oldname* into *newname*

push dword inputfile

push dword outputfile

call [rename]

add esp, 4\*2

cmp eax, 0 ; returns 0 if it is successful. On an error, the function returns a nonzero value

je theend ; and an error message which can be printed using the “perror()” function

; call perror(eroare) in case of error so that we see a more detailed error message.

push dword eroare

call [perror]

add esp, 4\*1

theend:

; exit(0)

push dword 0

call [exit]

The C source for the first part of the program is:

#include<stdio.h>

int main() {

FILE \* fd\_in, \*fd\_out;

char c;

int count=0;

fd\_in = fopen("file.txt", "r");

if (fd\_in>0){

fd\_out = fopen("out.txt", "w");

if (fd\_out>0){

do{

count=fread(&c, sizeof(char), 1, fd\_in);

c=c+1;

fwrite(&c, sizeof(char), 1, fd\_out);

}while (count>0);

fclose(fd\_out);

}

else

perror("Error Output file:");

fclose(fd\_in);

}

else

perror("Error Input file:");

exit(0);

}