ASM Project 2023

Exercises

1. Instructions

This project is individual and each student should provide an archive for their result.

Each exercise is independent and will be noted as such. All exercise must be implemented in assembly x86 64bits. Call to external functions are **forbidden** except if explicitly allowed, the presence of external symbols in your code will automatically result in a 0 for the exercise.

The result should be a zip archive which contains a folder for each exercise with the assembly file in it. The name of the folder and of the assembly file is given for each exercise. At the root of the zip file there should be a file AUTHOR containing their EPITA email address.

The final zip file should be named asm2023.zip and must have the following architecture:

```
asm2022/AUTHOR
asm2022/myadd/
asm2022/myadd/myadd.S
asm2022/fibonnaci/
...
```

The archive should be sent by mail at bruno.pujos@epita.fr with as object [ASM2023] EMAIL@epita.fr with EMAIL replace by the email of the student at latest February 10th 2023.

The software used are:

- nasm
- gcc

Exercise will be compiled with:

```
nasm -f elf64 FILENAME.S
```

2. Exercises

Exercise: myadd (1 point)

File: myadd/myadd.S

Write a function myadd which has the following prototype:

```
unsigned long long myadd(unsigned long long, unsigned long long);
```

The function should make the addition of the two arguments and return it.

The file *myadd.c* can be used for performing tests.

Exercise: fibonnaci (2 points)

File: fibonnaci/fibonnaci.S

Write a function *fibonnaci* in assembly which takes one argument, with the following prototypes:

```
long long fibonnaci(long long);
```

The function should calculate the fibonnaci number. If a negative number is given in argument, the function should always return -1.

The file *fibonnaci.c* can be used for performing tests.

Exercise: mymul (2.5 points)

File: mymul/mymul.S

Write a function *mymul* in assembly which takes three arguments, with the following prototypes:

```
typedef struct {
        unsigned long long high;
        unsigned long long low;
} resmul;

void mymul(unsigned long long, unsigned long long, resmul *res);
```

It should perform a multiplication of the first argument and the second argument and return the result in the res structure.

The file *mymul.c* can be used for performing tests.

Exercise: mydiv (2.5 points)

File: mydiv/mydiv.S

Write a function mydiv in assembly which takes 3 arguments, with the following prototypes:

```
typedef struct {
  unsigned long long res;
```

```
    unsigned long long rest;
} resdiv;

char mydiv(unsigned long long val, unsigned long long div, resdiv *res);
```

The function should divide val by div (val/div) and put the result in the resdiv structure. The resdiv.res should contain the result of the division, and the resdiv.rest parameter should have the result of the modulo. The function should return 1 on success and 0 if a division by 0 has been detected.

The file *mydiv.c* can be used for performing tests.

Exercise: myitoa (2.5 points)

File: myitoa/myitoa.S

Write a function *myitoa* in assembly which takes one argument, with the following prototypes:

```
long long myitoa(char *s);
```

The function should transform the string given in argument to an integer. The negative number should be handled.

The file *myitoa.c* can be used for performing tests.

Exercise: myfilexor (4.5 points)

File: myfilexor/myfilexor.S

Write a function *myfilexor* in assembly which has 3 arguments, with the following prototypes:

```
char *myfilexor(char *filepath, char key);
```

The function should read the content of the file, xor each byte of its content with the key and return the resulting allocated string. Call to external functions in libc are allowed.

The file *myfilexor.c* and *filetoxor.bin* can be used for performing tests.

Exercise: myorderlist (5 points)

File: myorderlist/myorderlist.S

Write a function *orderlist* in assembly which takes 2 arguments, with the following prototypes:

```
void orderlist(long long *list, int size);
```

The function should print on the standard output (STDOUT) the list in order in decimal, separated by comma followed by a space and finish with a \n and no space at the end of the line. The biggest number to be given in the list will be 0xFFFFFFFF, the smallest one 0. The maximum size of the list (argument size) will be 0x200 and minimum 0x1.

Call to external functions are NOT allowed. You should use syscall for printing on the standard output and allocating memory (mmap).

The file *myorderlist.c* can be used for performing tests.