x86-64 Assembly Exam Retake

Table of Contents

1. Preamble

Before starting the exam, understand the following:

- Make sure you have **read** and **understood** the guidelines on Moodle Exam.
- Make sure to read the whole subject before starting.
- **Do not forget to review your submission** before leaving the exam, you can use the Safety checklist. The review time is included in the overall time: reserve about 20-30 minutes.

1.1. Safety checklist

Please double check your submission.

- `[ ]` It has been uploaded on Moodle Exam.
- `[ ]` The submission file structure is the one given in the subject.
- `[ ]` Run all the tests and make sure they do not fail. You may try to tweak compilation flags.

1.2. Testing

Basic tests suites may be provided for each exercises. They are here to help you make progress and will prevent you to spend too much time on debugging. However, it is your responsibility to make sure the code work using the principles seen in class.

You are encouraged to add your own tests. However they will not be evaluated as the provided files will be replaced during evaluation. Tests are subjected to change and may add other test cases. Make sure to check the corner cases if any.

1.3. Tooling

Programs such as `gdb`, `objdump`, `readelf`, `nm` are allowed.

Any usage of `gcc` as a code generator for assembly is **prohibited** and will be checked in your submission. Exercises submitted with generated code will be **voided** and noticed.

1.3.1. Helpful gdb setup using `~/.gdbinit`

In order to have comfortable conditions to debug assembly code, you may use the following `${HOME}/.gdbinit` file in your home directory.

```
tui new-layout debugasm {-horizontal asm 1 regs 1} 2 cmd 1
tui enable
```

As a reminder, commands to debug are `start`, `layout debugasm`, `stepi`, `nexti`, `finish`. The complete gdb manual is available through `info` pages.

```
42sh$ info gdb
```

2. Extract given files

An archive is provided on Moodle Exam with given files that may contain a basic tests suite used to evaluate assingments. In order to prepare your submission directory, you may want to extract the archive inside a desired file structure. Refer to submission instructions.

```
# E[x]tract a (compressed) archive [f]ile into the current directory [v]erbosely:
42sh$ tar xvf path/to/given-files.tgz
```

3. Submission

3.1. Moodle archive tgz format

The submission will be under the `.tgz` format. It's an archive format similar to `.zip`. Make sure to submit **a clean archive**[1] with the given structure for your submission on Moodle Exam as your grade will be computed automatically.

A `.tgz` archive can be achieved with the following command[2]:

```
# [c]reate a compressed archive and write it to a [f]ile, using [a]rchive
# suffix to determine the compression program:
42sh$ tar caf path/to/target.tgz path/to/file1 path/to/file2 ...

# Lis[t] the contents of a tar [f]ile [v]erbosely:
42sh$ tar tvf path/to/source.tgz
```

Make sure to follow the submission's guidelines.

3.2. Directory Structure

The following example is the expected directory structure obtained using the `tree(1)` command. The archive's root must contain a directory named `asm-xavier.loginard`. See Login.

```
asm-xavier.loginard/
|-- fact/
|   |-- Makefile
|   |-- fact.s
|   `-- fact_tests.cpp
|-- includes/
|   `-- doctest.h
|-- memcmp_rep/
|   |-- Makefile
|   |-- memcmp.s
|   `-- memcmp_tests.cpp
|-- strcpy/
|   |-- Makefile
|   |-- strcpy.s
|   `-- strcpy_tests.cpp
`-- write_hello/
    |-- Makefile
    |-- write_hello.S
    `-- write_hello_tests.cpp
```

3.2.1. Login

In the above example, the student's login is `xavier.loginard`. In order to match your identity with the ID of Moodle, replace `xavier.loginard` with **your** complete login.

Wrong logins will not be fixed.

- Replace `xavier.loginard` with your login.
- The archive name **must** be `asm-xavier.loginard.tgz`.

4. MCQ (6 pts)

Complete the MCQ on Moodle Exam.

5. Exercises

5.1. Fact (2 pts)

Write an assembly function `fact` that implements the mathematical *factorial*. The factorial of a non-negative integer n, denoted by $n!$, is the product of all positive integers less than or equal to n.

$$
\begin{aligned}
0! &= 1 \\
n! &= n \cdot (n-1)! \qquad \text{when } n \geq 1
\end{aligned}
$$

Usage of any lib C functions is prohibited.

```
unsigned int fact(unsigned int n);
```

5.1.1. Expected results

```
make -B fact_tests && ./fact_tests -s
```

```
g++ -O2 -I ../includes/ -m64 -c -o fact_tests.o fact_tests.cpp
as  -64 -o fact.o fact.s
gcc -lstdc++ -lm -z noexecstack -m64 fact_tests.o fact.o   -o fact_tests
[doctest] doctest version is "2.4.8"
[doctest] run with "--help" for options
===============================================================================
fact_tests.cpp:8:
TEST CASE:  testing the factorial function

fact_tests.cpp:9: SUCCESS: CHECK( fact(0) == 1 ) is correct!
  values: CHECK( 1 == 1 )

fact_tests.cpp:10: SUCCESS: CHECK( fact(1) == 1 ) is correct!
  values: CHECK( 1 == 1 )

fact_tests.cpp:11: SUCCESS: CHECK( fact(2) == 2 ) is correct!
  values: CHECK( 2 == 2 )

fact_tests.cpp:12: SUCCESS: CHECK( fact(3) == 6 ) is correct!
  values: CHECK( 6 == 6 )

fact_tests.cpp:13: SUCCESS: CHECK( fact(10) == 3628800 ) is correct!
  values: CHECK( 3628800 == 3628800 )

===============================================================================
[doctest] test cases: 1 | 1 passed | 0 failed | 0 skipped
[doctest] assertions: 5 | 5 passed | 0 failed |
[doctest] Status: SUCCESS!
```

5.2. String copy (2 pts)

This function must have the same behavior as the `strcpy` function described in the third section of the man `strcpy(3)`.

Usage of any lib C functions is prohibited.

```
char *strcpy(char *dest, const char *source);
```

5.2.1. Expectations

```
make -B strcpy_tests && ./strcpy_tests -s
```

```
g++ -O2 -I ../includes -m64 -c -o strcpy_tests.o strcpy_tests.cpp
as   --64 -o strcpy.o strcpy.s
gcc -lstdc++ -lm -z noexecstack -m64 strcpy_tests.o strcpy.o   -o strcpy_tests
[doctest] doctest version is "2.4.8"
[doctest] run with "--help" for options
===============================================================================
strcpy_tests.cpp:9:
TEST CASE:  Small string copy

strcpy_tests.cpp:12: SUCCESS: CHECK( strcpy(dst_str1, "Hello World!") == (char *)dst_str1 ) is
  values: CHECK( 0x00007ffc00bcae00 == 0x00007ffc00bcae00 )

strcpy_tests.cpp:13: SUCCESS: CHECK( std::string{dst_str1} == "Hello World!" ) is correct!
  values: CHECK( Hello World! == Hello World! )
```

```
===============================================================================
[doctest] test cases: 1 | 1 passed | 0 failed | 0 skipped
[doctest] assertions: 2 | 2 passed | 0 failed |
[doctest] Status: SUCCESS!
```

5.3. String compare (5 pts)

This function must have the same behavior as the `memcmp` function described in the third section of the man `memcmp(3)`[3].

The implementation **must** use `rep/repe/repz/repne/repnz` prefixes (refer to the documentations, including the GNU as one). Using a regular loop **will not** give points.

Usage of any lib C functions is prohibited.

For this exercise, there is no need to specify the `%es` segment.

```
int my_memcmp(const void *s1, const void *s2, size_t n);
```

5.3.1. Expectations

```
make -B memcmp_tests && ./memcmp_tests -s
```

```
g++ -O2 -I ../includes -m64 -c -o memcmp_tests.o memcmp_tests.cpp
as   --64 -o memcmp.o memcmp.s
gcc -lstdc++ -lm -z noexecstack -m64 memcmp_tests.o memcmp.o   -o memcmp_tests
[doctest] doctest version is "2.4.8"
[doctest] run with "--help" for options
===============================================================================
memcmp_tests.cpp:9:
TEST CASE:  Compare strings

memcmp_tests.cpp:10: SUCCESS: CHECK( my_memcmp("Hello World!", "Hello World!", 12) == 0 ) is co
  values: CHECK( 0 == 0 )

memcmp_tests.cpp:11: SUCCESS: CHECK( my_memcmp("Hello World!", "No match", 5) < 0 ) is correct!
  values: CHECK( -6 <  0 )

===============================================================================
[doctest] test cases: 1 | 1 passed | 0 failed | 0 skipped
[doctest] assertions: 2 | 2 passed | 0 failed |
[doctest] Status: SUCCESS!
```

5.4. Syscalls (5 pts)

```
void write_hello_world(void);
```

The function `write_hello_world()` creates a file using only syscalls. The created file satisfies the following conditions:

- The filename is fixed and must be written in the current directory `./hello` with the tests.

Usage of any lib C functions is prohibited. You do not need the `call` instruction for this exercise.

5.4.1. File output examples

Calling `write_hello_world();` will create the file `hello` which contains:

```
Hello World!
```

5.4.2. Syscall information

You will need to call `open(2)`, `write(2)` and `close(2)`. The system calling convention for syscalls requires the syscall number to call.

The number is defined in the `sys/syscall.h` header. You can include the header by inserting the following line:

```
#include <sys/syscall.h>
```

Syscalls values are replaced by the macros `SYS_open`, `SYS_write`, `SYS_close`. Use an immediate value in assembly.

Refer to the man page `(2)` for their documentation.

1. Call to `open(2)`
   - The value for the **flags** `O_WRONLY | O_CREAT | O_TRUNC` is `0x241`.
   - The value for the **mode** is `400` as we want the user to be able to read the file.
   1. Expectations

      ```
      make -B write_hello_tests && ./write_hello_tests -s
      ```

      ```
      g++ -O2 -I ../includes -m64 -c -o write_hello_tests.o write_hello_tests.cpp
      gcc  -I ../includes  -c -o write_hello.o write_hello.S
      gcc -lstdc++ -lm -z noexecstack -m64 write_hello_tests.o write_hello.o   -o write_he
      [doctest] doctest version is "2.4.8"
      [doctest] run with "--help" for options
      ===============================================================================
      write_hello_tests.cpp:10:
      TEST CASE:  Check hello world

      write_hello_tests.cpp:16: SUCCESS: REQUIRE( f.is_open() ) is correct!
        values: REQUIRE( true )

      write_hello_tests.cpp:20: SUCCESS: CHECK( f.tellg() == 12 ) is correct!
        values: CHECK( 12 == 12 )

      write_hello_tests.cpp:27: SUCCESS: CHECK( content == "Hello World!" ) is correct!
        values: CHECK( Hello World! == Hello World! )

      ===============================================================================
      [doctest] test cases: 1 | 1 passed | 0 failed | 0 skipped
      [doctest] assertions: 3 | 3 passed | 0 failed |
      [doctest] Status: SUCCESS!
      ```

Footnotes:

[1] : If you are using `git`, you can export a git tree to an archive using git-archive. See `man 1 git-archive` and the `--prefix` option.

[2] : `man 1 tar`

[3] : Read it carefully.