DLDCA Lab 2: Assembly
Week 2

August 26, 2025

Introduction

In this lab we are going to start with a simple set of problems through which we will understand the basics of x86 assembly syntax and semantics.

Necessary tools

We shall be using `nasm` in order to compile x86 assembly code.

Before starting the lab, ensure that the tool is installed in your system. This can be verified by running the following:

```
$ nasm
nasm: fatal: no input file specified
Type nasm -h for help.
```

Structure of submission/templates

```
your-roll-no/
|- task1/
    |- loops.asm (TODO)
    |- compile.sh
|- task2/
    |- tower-of-hanoi.asm (TODO)
    |- compile.sh
```

The folder structure of the expected submission for this lab is given above. We expect this folder to be compressed into a tarball with the naming convention of your-roll-no.tar.gz. The command given below can be used to do the same

```
$ tar -cvzf your-roll-no.tar.gz your-roll-no/
```

Tasks

An overview of tasks can be seen in `TODO.md`. **Please go through this file before starting the lab**.

Task 1: Loops in x86 assembly

The goal of this task is to learn about how loops work in x86 assembly.

To do this, you should go through the following 3 step procedure:

1. Take in a number, $n > 0$ as input from user, using `read(0,input_buf,20)`
2. Find the n^{th} fibonacci number, using the rule that the n^{th} fibonacci number is given by the sum of the $(n-1)^{th}$ fibonacci number and the $(n-2)^{th}$ fibonacci number, where we define the 0^{th} fibonacci number to be 0 and the 1^{st} fibonacci number to be 1
3. Print the n^{th} fibonacci number to the screen by converting it to a string and then using the `write` system call

The syscall number for the read syscall is 0, while the number for the write syscall is 1.

A C++ for loop that looks like:

```
for (<initialization>; <conditional>; <update>) {
<code>
}
```

Can be written in assembly along the lines of:

```
<initialization>
.forBegin:
<conditional>
if <conditional> jmp .forEnd
```

```
<code>
<update>
.forEnd:
```

To run your code, you may run:

```
$ bash compile.sh
$ ./loops
```

OR you can do it manually using,

```
nasm -f elf64 loops.asm -o loops.o
ld loops.o -o loops
./loops
```

Some expected outputs are:

```
$ ./loops
5
5$
```

```
$ ./loops
7
13$
```

Note there is no newline after the output.

Task 2: Tower of Hanoi in ASM

The goal of this task is to learn about how to implement functions in x86 assembly.

You have to implement the following functions:

1. `printNum`: Given an arbitary 64-bit number stored in `rax`, it prints the result to stdout
2. `printFromAndTo`: Given 2 strings whose memory addresses are stored in `rax` and `rdi`, this function should print `" from *rax to *rdi"` to stdout
3. `hanoi`: Given a C function, you have to translate it to assembly. Use the aforementioned functions to print to stdout, you may also print to stdout using your own code in this function

4. `_start`: Take in a number, n as input using the `read` syscall, and solve the tower of hanoi puzzle for n disks by calling the `hanoi` function

Please stick to convention of caller and callee saved registers. **Important: Before calling syscalls, ensure your stack pointer is a multiple of 16, otherwise you may run into errors.**

To run your code, you may run:

```
$ bash compile.sh
$ ./tower-of-hanoi
```

OR you can do it manually using,

```
nasm -f elf64 tower-of-hanoi.asm -o tower-of-hanoi.o
ld tower-of-hanoi.o -o tower-of-hanoi
./tower-of-hanoi
```

Some expected outputs are:

```
$ ./tower-of-hanoi
5
Shifted disk 1 from A to B
Shifted disk 2 from A to C
Shifted disk 1 from B to C
Shifted disk 3 from A to B
Shifted disk 1 from C to A
Shifted disk 2 from C to B
Shifted disk 1 from A to B
Shifted disk 4 from A to C
Shifted disk 1 from B to C
Shifted disk 2 from B to A
Shifted disk 1 from C to A
Shifted disk 3 from B to C
Shifted disk 1 from A to B
Shifted disk 2 from A to C
Shifted disk 1 from B to C
Shifted disk 5 from A to B
Shifted disk 1 from C to A
Shifted disk 2 from C to B
Shifted disk 1 from A to B
Shifted disk 3 from C to A
Shifted disk 1 from B to C
Shifted disk 2 from B to A
```

```
Shifted disk 1 from C to A
Shifted disk 4 from C to B
Shifted disk 1 from A to B
Shifted disk 2 from A to C
Shifted disk 1 from B to C
Shifted disk 3 from A to B
Shifted disk 1 from C to A
Shifted disk 2 from C to B
Shifted disk 1 from A to B
$
```

```
$ ./tower-of-hanoi
3
Shifted disk 1 from A to B
Shifted disk 2 from A to C
Shifted disk 1 from B to C
Shifted disk 3 from A to B
Shifted disk 1 from C to A
Shifted disk 2 from C to B
Shifted disk 1 from A to B
$
```

You are encouraged to use the assembly instructions cheatsheet which has been given to you to understand the various instructions that may be used to implement conditional statements.

Assembly instructions cheatsheet (non-exhaustive)

Instruction	Operands	Effect
mov	reg, reg/∗mem/imm	reg ← reg / ∗mem / imm
mov	∗mem, reg	∗mem ← reg
add	reg, reg/∗mem/imm	reg ← reg + reg/∗mem/imm
add	∗mem, reg/imm	∗mem ← ∗mem + reg/imm
sub	reg, reg/∗mem/imm	reg ← reg − reg/∗mem/imm
sub	∗mem, reg/imm	∗mem ← ∗mem − reg/imm
xor	reg, reg/∗mem/imm	reg ← reg ⊕ reg/∗mem/imm
xor	∗mem, reg/imm	∗mem ← ∗mem ⊕ reg/imm
and	reg, reg/∗mem/imm	reg ← reg ∧ reg/∗mem/imm
and	∗mem, reg/imm	∗mem ← ∗mem ∧ reg/imm
or	reg, reg/∗mem/imm	reg ← reg ∨ reg/∗mem/imm
or	∗mem, reg/imm	∗mem ← ∗mem ∨ reg/imm
imul	reg, reg/∗mem/imm	reg ← reg × reg/∗mem/imm
imul	∗mem, reg/imm	∗mem ← ∗mem × reg/imm
div	reg	rax ← (rdx:rax) / reg rdx ← (rdx:rax) % reg
cmp	reg/∗mem, reg/imm	Sets flags based on subtraction: reg/∗mem − reg/imm. ZF = 1 if equal, SF = 1 if result negative, CF = 1 if borrow (unsigned), OF = 1 if signed overflow.
test	reg/∗mem, reg/imm	Sets flags based on bitwise AND: reg/∗mem ∧ reg/imm. ZF = 1 if result is zero, SF = 1 if high bit set, CF and OF are cleared. No result is stored.
inc	reg/∗mem	reg/∗mem ← reg/∗mem + 1
dec	reg/∗mem	reg/∗mem ← reg/∗mem − 1
jmp	label	Unconditional jump to label
je / jz	label	Jump if Zero Flag (ZF) = 1
jne / jnz	label	Jump if Zero Flag (ZF) = 0
jl	label	Jump if Less (SF ≠ OF)
jle	label	Jump if Less or Equal (ZF = 1 or SF ≠ OF)
jg	label	Jump if Greater (ZF = 0 and SF = OF)
jge	label	Jump if Greater or Equal (SF = OF)
syscall		Performs a syscall with syscall number stored in rax Arguments passed in other registers starting from rdi
push	reg	Pushes the value in reg onto the stack
pop	reg	Pops the value on top of the stack onto reg
call	label	Calls the function called label

Table 1: Common x86-64 Instructions and Their Effects