

# CSE-3113: Microprocessor Lab Course (Lab-1)

## Using NASM to Perform Arithmetic Operations and Call printf in Assembly

Jubair Ahammad Akter

### 1. Objective

To write and execute a NASM program that adds two 64-bit integers, stores the result in memory, and prints the output using the C `printf()` function.

### 2. Prerequisites and Environment Setup

Before running the NASM program, the following setup steps are required in an Ubuntu (Linux) terminal.

#### Installation and Compilation Commands

```
sudo apt install nasm -y
nasm -v

nasm -f elf64 hello.asm -o hello.o
gcc -no-pie hello.o -o hello
./hello
```

#### Explanation of Each Command:

- `sudo apt install nasm -y` — Installs the NASM assembler on the system using the package manager. The `-y` flag automatically confirms installation.
- `nasm -v` — Displays the currently installed NASM version to confirm successful setup.
- `nasm -f elf64 hello.asm -o hello.o` — Assembles the source file `hello.asm` into an object file `hello.o` using the 64-bit ELF format (Linux binary format).
- `gcc -no-pie hello.o -o hello` — Links the assembled object file with the C standard library using GCC. The `-no-pie` flag ensures the output is a non-Position Independent Executable (traditional memory layout).
- `./hello` — Executes the final binary and displays the program output.

```
adibnt@DESKTOP-MC1CBMA: ~$ nasm -v
Command 'nasm' not found, but can be installed with:
sudo apt install nasm
adibnt@DESKTOP-MC1CBMA: ~$ sudo apt install nasm
[sudo] password for adibnt:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following NEW packages will be installed:
  nasm
0 upgraded, 1 newly installed, 0 to remove and 28 not upgraded.
Need to get 375 kB of archives.
After this operation, 3345 kB of additional disk space will be used.
Get:1 http://archive.ubuntu.com/ubuntu jammy/universe amd64 nasm amd64 2.15.05-1 [375 kB]
Fetched 375 kB in 2s (190 kB/s)
Selecting previously unselected package nasm.
(Reading database ... 50391 files and directories currently installed.)
Preparing to unpack ../nasm_2.15.05-1_amd64.deb ...
Unpacking nasm (2.15.05-1) ...
Setting up nasm (2.15.05-1) ...
Processing triggers for man-db (2.10.2-1) ...
adibnt@DESKTOP-MC1CBMA: ~$ nasm -v
NASM version 2.15.05
adibnt@DESKTOP-MC1CBMA: ~$ |
```

Figure 1: Installing the NASM assembler and check version

```
adibnt@DESKTOP-MC1CBMA: /mnt/j/JUBAIR AHAMMAD AKTER/29-3-1/LAB3_MProcessor/Lab1$ nasm -f win64 hello.asm -o hello.obj
adibnt@DESKTOP-MC1CBMA: /mnt/j/JUBAIR AHAMMAD AKTER/29-3-1/LAB3_MProcessor/Lab1$ gcc -no-pie hello.o -o hello
adibnt@DESKTOP-MC1CBMA: /mnt/j/JUBAIR AHAMMAD AKTER/29-3-1/LAB3_MProcessor/Lab1$ ./hello
a=5, b=2 c=7
adibnt@DESKTOP-MC1CBMA: /mnt/j/JUBAIR AHAMMAD AKTER/29-3-1/LAB3_MProcessor/Lab1$ |
```

Figure 2: Assembling source file object file, linking with C using GCC, executing the final binary file and displaying program output

### 3. NASM Code: Addition and printf Demonstration

Filename: hello.asm

```
extern printf

SECTION .data
a:      dq 5
b:      dq 2
c:      dq 0
fmt:    db "a=%ld, b=%ld, c=%ld", 10, 0

SECTION .text
global main

main:
    push    rbp

    mov     rax, [a]      ; Load a = 5 into RAX
    mov     rbx, [b]      ; Load b = 2 into RBX
    add     rax, rbx      ; RAX = a + b = 7
    mov     [c], rax      ; Store result in c

    mov     rdi, fmt      ; 1st argument: format string
    mov     rsi, [a]      ; 2nd argument: a
    mov     rdx, [b]      ; 3rd argument: b
    mov     rcx, [c]      ; 4th argument: c
    mov     rax, 0        ; No vector registers used
    call    printf        ; Call printf to print values

    pop     rbp
    mov     rax, 0
    ret
```

## 4. Explanation

### Line-by-Line Explanation

- **extern printf** — Declares an external C function to be linked later by `gcc`.
- **SECTION .data** — Contains initialized data (variables and strings).
- **dq** — Defines a 64-bit (quadword) integer.
- **fmt** — The format string for `printf()`, ending with a newline and null terminator.
- **SECTION .text** — Marks the start of the code section.
- **global main** — Defines the program entry point.
- **mov/add instructions** — Perform arithmetic using registers `RAX` and `RBX`.
- **mov [c], rax** — Saves the sum into memory variable `c`.
- **RDI, RSI, RDX, RCX** — Hold the first four arguments to `printf()` (System V ABI).
- **mov rax, 0** — Required before calling a variadic function.
- **call printf** — Prints: `a=5, b=2, c=7`.
- **ret** — Returns control to the operating system.

## 5. Conclusion

In this lab, a simple NASM program was developed to demonstrate arithmetic operations, memory manipulation, and interaction with C library functions using the System V AMD64 calling convention.