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.

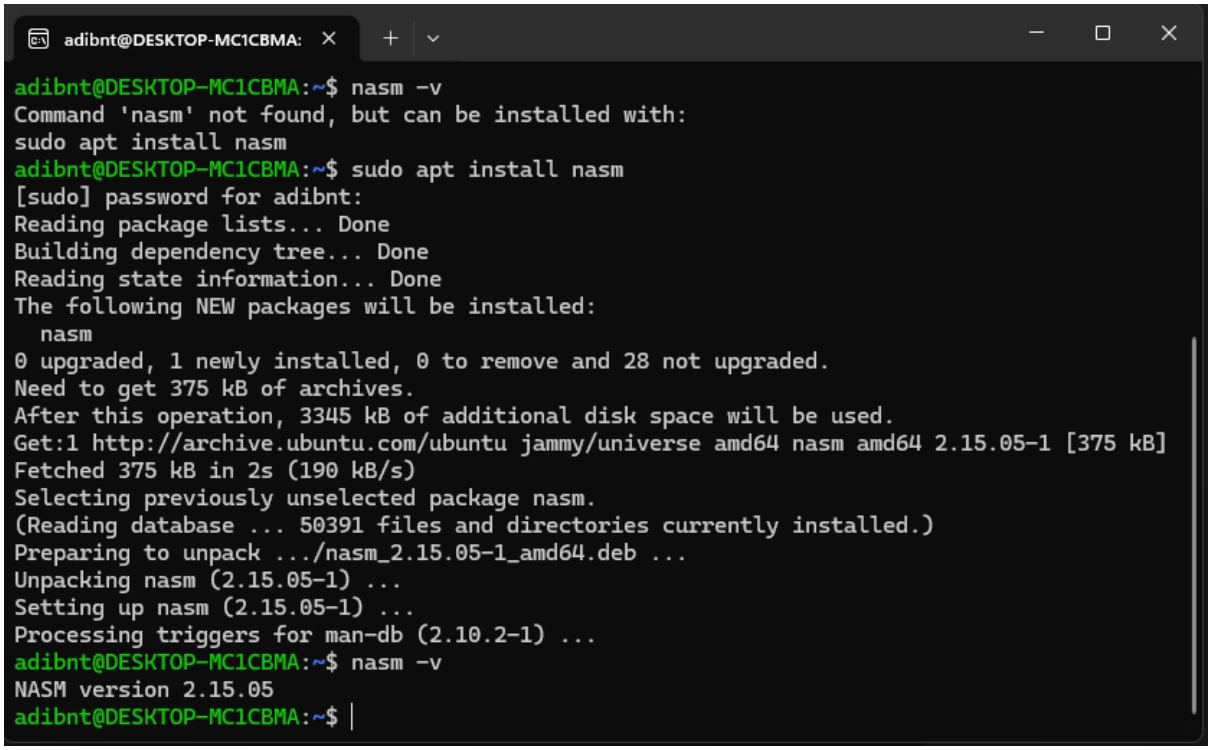

Figure 1: Installing the NASM assembler and check version

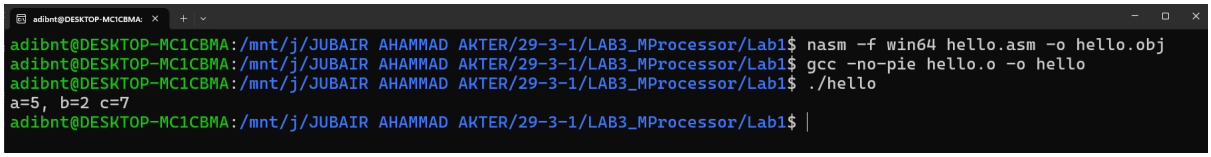

Figure 2: Assembling source file object file, linking with C using GCC, executing the final binary file and displaying program output

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.