CSE-3113: Microprocessor Lab Course (Lab - 2)

Topic: Using NASM with scanf and printf for Arithmetic Operations

Jubair Ahammad Akter

Section 1: NASM Source Code

Program Code

```
extern  printf
extern  scanf

SECTION .data

a:      dq  5
b:      dq  2
c:      dq  0

enter:      db "Enter two numbers: ",0
out_fmt:    db "%ld + %ld = %ld", 10, 0
out_fmt_2:  db "%s", 10, 0
in_fmt:     db "%d", 0

SECTION .text

global main
main:
    push    rbp

    mov     rax, 0
    mov     rdi, out_fmt_2
    mov     rsi, enter
    call    printf

    mov     rax, 0
    mov     rdi, in_fmt
    mov     rsi, a
    call    scanf
```

Program Code

```
    mov     rax, 0
    mov     rdi, in_fmt
    mov     rsi, b
    call    scanf

    mov     rax, [a]
    mov     rbx, [b]
    add     rax, rbx
    mov     [c], rax

    mov     rdi, out_fmt
    mov     rsi, [a]
    mov     rdx, [b]
    mov     rcx, [c]
    mov     rax, 0
    call    printf

    pop     rbp
    mov     rax, 0
    ret
```

Section 2: Line-by-Line Explanation

1. External Function Declarations

- `extern printf` — Declares that the C library function `printf()` will be used. The actual implementation is linked at compile time by `gcc`.
- `extern scanf` — Similarly declares the external `scanf()` function for input.

2. Data Section

- `SECTION .data` — Marks the start of the initialized data segment, which stores global variables and strings.
- `a: dq 5, b: dq 2, c: dq 0` Each variable reserves 8 bytes (a **quadword**).
  - `dq` means "Define Quadword" — allocates 8 bytes for a 64-bit integer.

  - So, `a`, `b`, and `c` are 64-bit memory locations holding integer values.

- `enter: db "Enter two numbers:", 0` Defines a string used as a prompt. `db` stands for "Define Byte" — it stores text or characters one byte at a time, ending with a null byte (0) for C compatibility.
- `out_fmt: db "%ld + %ld = %ld", 10, 0` This is a format string for `printf`. `10` is ASCII for newline ('\n'), and the final `0` marks the end of the string.
- `out_fmt_2: db "%s", 10, 0` Format string to print another string (used to display the prompt).
- `in_fmt: db "%d", 0` Format string for `scanf`, specifying that an integer will be read.

3. Text Section and Entry Point

- `SECTION .text` — Marks the code segment where all executable instructions reside.
- `global main` — Makes the `main` label visible to the linker so that execution starts from here.

4. Function Prologue

- `push rbp` — Saves the current base pointer to the stack for stack frame setup.

5. Printing the Prompt

- `mov rax, 0` — Required before calling a variadic function like `printf()` (indicates no vector registers are used).
- `mov rdi, out_fmt_2` — The first argument: format string "%s".
- `mov rsi, enter` — The second argument: string to print.
- `call printf` — Prints "Enter two numbers:" on the screen.

6. Reading the First Number

- `mov rax, 0` — Reset before variadic call.
- `mov rdi, in_fmt` — Format string "%d".
- `mov rsi, a` — Address where the input will be stored.
- `call scanf` — Reads the first integer into memory location `a`.

7. Reading the Second Number

- Same as before, but stores input into `b`.

8. Performing Addition

- `mov rax, [a]` — Loads the first number into register RAX.
- `mov rbx, [b]` — Loads the second number into RBX.
- `add rax, rbx` — Adds the two values (RAX = a + b).
- `mov [c], rax` — Stores the result back into memory location `c`.

9. Displaying the Result

- `mov rdi, out_fmt` — Format string "%ld + %ld = %ld".
- `mov rsi, [a]` — First value (a).
- `mov rdx, [b]` — Second value (b).
- `mov rcx, [c]` — Result (c).
- `mov rax, 0` — Reset before `printf`.
- `call printf` — Displays the result, e.g., "10 + 20 = 30".

10. Function Epilogue

- `pop rbp` — Restores the previous base pointer.
- `mov rax, 0` — Sets return value of `main` (0 = success).
- `ret` — Returns control to the operating system.

Program Output Example

Sample Terminal Output

```
Enter two numbers:
10
20
10 + 20 = 30
```

Theory Summary

- **db (Define Byte):** Allocates 1 byte per value. Used for strings and characters.
- **dq (Define Quadword):** Allocates 8 bytes (64 bits). Used for integers or pointers in 64-bit mode.
- **Registers Used:**
  - `RAX` — General-purpose register for calculations and function calls.
  - `RBX` — Secondary register for arithmetic.
  - `RDI, RSI, RDX, RCX` — Used to pass the first four function arguments (System V AMD64 calling convention).
- **Calling Convention:** In Linux x86-64, function arguments are passed in the following registers (in order): `RDI, RSI, RDX, RCX, R8, R9`.