# Contents

# 1 Lecture Notes on x86-64 Assembly Basics

Welcome to today's lecture on x86-64 assembly basics. We'll cover various fundamental aspects, including register usage, memory addressing, and instruction sets, focusing specifically on move and LEAQ instructions. This lecture is essential for understanding low-level programming and is fundamental for courses like operating systems or computer architecture.

## 1.1 Introduction to x86-64 Assembly

x86-64 is an extension of the x86 instruction set that supports 64-bit computing, offering enhanced performance through a broader set of registers and more extensive memory addressing capabilities.

### 1.1.1 Registers Overview

- Registers are small, high-speed storage locations directly within the CPU used to store temporary data.
- In x86-64, there are 16 general-purpose registers named RAX, RBX, RCX, RDX, RSI, RDI, RBP, RSP, and R8 to R15.
- **RAX, RBX, RCX, and RDX** are the primary general-purpose registers with historical significance and specific use cases.
- **RSI** and **RDI** are typically used for source and destination indices in string operations.
- **RBP** and **RSP** are used for base and stack pointers, crucial for function call management.
- Registers can be accessed entirely (64-bit), or in parts (32-bit as EAX or 8-bit AL), to support backward compatibility with older 32-bit and 8-bit software.

### 1.1.2 Register Naming Conventions

Register naming conventions follow a pattern where "R" indicates a 64-bit register (e.g., RAX), "E" indicates a 32-bit register (e.g., EAX), and no prefix with "H" or "L" suffixes indicates high or low 8-bit parts of the register (e.g., AH or AL).

### 1.1.3 Memory Addressing Modes

Memory addressing in x86-64 allows for various modes to access data:

1. **Immediate Addressing:** Uses a constant value within the instruction. e.g., `mov $5, %rax`

2. **Register Addressing:** Uses the content of a register as the operand. e.g., `mov %rax, %rbx`

3. **Memory Addressing:** Accesses data in memory through complex addressing modes involving registers and constants.

#### 1.1.4 Instruction Set Overview

- The **x86-64 instruction set** encompasses a wide range of operations, including arithmetic, logic, control flow, and data movement.
- **Move Instruction (mov/movq):** Transfers data between registers, memory and immediate values. It's akin to copying rather than moving in high-level programming.
- **Load Effective Address (LEAQ):** Computes the effective address of the operand and stores it in a register without accessing the memory. This instruction can perform arithmetic operations like addition or multiplication by powers of two without executing arithmetic instructions.

#### 1.1.5 Examples and Explanation

**Move Instruction**

- **MoveQ** is used to move 64-bit data and has the syntax `move source, destination`.
- **Immediate to Register:** `movq $5, %rax` moves the constant 5 to RAX.
- **Register to Memory:** `movq %rax, (%rbx)` moves RAX's value to the memory address contained in RBX.

**LEA Instruction**

- **LEAQ** is particularly useful for calculating addresses or simple arithmetic without accessing memory.
- Example: `leaq (%rdi,%rsi,4), %rax` computes the address that is the sum of RDI's content and RSI's content times four, storing the result in RAX.

### 1.2 Practical Application and Memory Addressing

- **Memory Operations:** In x86-64, operations directly involving memory are powerful but require careful handling of addressing modes.
- **Addressing Modes:** The combination of base registers, index registers, scaling factors, and displacements offers flexibility in generating addresses dynamically during program execution.

- **Use of LEAQ:** Beyond calculating addresses, LEAQ can be used for several arithmetic operations like multiplying a register value by constants (e.g., `leaq 0(,%rdi,4), %rax` multiplies RDI by four).

### 1.2.1 Closing Tips for Assembly Programming

- **Understand the Register Usage:** Knowing which registers to use for specific tasks can optimize your assembly code.
- **Master the Addressing Modes:** Complex memory operations often hinge on effectively using the various addressing modes.
- **Limit Direct Memory Access:** If possible, use registers for intermediate calculations to reduce costly memory accesses.

# 2 Concluding Remarks

Today's lecture introduced you to the basics of x86-64 assembly, focusing on registers, addressing modes, and key instructions like move and LEA. Understanding these concepts is vital for efficient low-level programming and will serve as a foundation for exploring more complex assembly programming techniques.