# Computer Architecture Lecture Note Study Guide

This study guide provides a comprehensive summary of key concepts covered in the Computer Architecture lectures, encompassing fundamental organization, assembly language programming, arithmetic and memory operations, number representations, and instruction formats. Understanding these topics is crucial for grasping how software interacts with hardware and how computers process information efficiently. The course emphasizes the MIPS architecture as a practical example for learning these principles.

The foundational concepts of computer architecture involve understanding the different levels of software and hardware. High-level languages like C, Java, and Python are human-readable and machine-independent, while assembly code is low-level, human-readable, and machine-dependent. The lowest level is machine code, represented by 0s and 1s, which is directly executed by the CPU. Key components of a microprocessor (CPU) include the Arithmetic Logic Unit (ALU) for calculations, registers for data storage during execution, control unit for timing and data flow, memory for instructions and data, and Input/Output (I/O) for external communication.

Introduction to Assembly Language Programming focuses on the MIPS architecture, which is a Reduced Instruction Set Computer (RISC). MIPS assembly uses a 32-bit register set for frequently accessed data, with specific registers for temporary values ($t), saved variables ($s), function arguments ($a), and return values ($v). Basic instructions include `li` (load immediate), `la` (load address), `lw` (load word from memory to register), `sw` (store word from register to memory), `add`, `sub`, and `addi` (add immediate). The MARS simulator is a crucial tool for writing, assembling, debugging, and running MIPS assembly programs.

Arithmetic and Memory Operations delve into how data is manipulated and stored. MIPS arithmetic instructions typically use three operands: two sources and one destination, adhering to the "simplicity favors regularity" design principle. Unsigned addition and subtraction are available, but signed operations can lead to overflow if results exceed the allowed bit representation. Memory operations involve loading data from memory into registers (`lw`) and storing data from registers into memory (`sw`). MIPS uses byte-addressable memory, meaning each byte has a unique address, and word addresses increment by 4 since a word is 4 bytes. Memory alignment is critical, as word operations require addresses to be multiples of 4. Endianness (big-endian vs. little-endian) dictates how bytes are ordered within a word.

Signed Number Representations explain how positive and negative numbers are handled in binary. Two primary methods are Signed Magnitude and Two's Complement. Signed Magnitude uses the Most Significant Bit (MSB) as a sign bit, but it has issues with addition and two representations for zero. Two's Complement, however, is widely used in computers because it simplifies arithmetic operations and has a single representation for zero. The procedure to find the two's complement of a number involves inverting all bits and adding one. Sign extension is also important, where the sign bit is copied to higher-order bits when converting a number to a larger bit representation, preserving its value.

Floating-point arithmetic provides a way to represent numbers with greater precision and range than fixed-point representations. The IEEE 754 standard defines single-precision (32-bit) and double-precision (64-bit) formats. A floating-point number is represented as ± M × B^E, where M is the mantissa, B is the base (2 for binary), and E is the exponent. The standard uses a biased exponent to represent both positive and negative exponents, and an implicit leading 1 for the mantissa to gain an extra bit of precision. Floating-point addition is more complex than integer addition, requiring exponents to be aligned before adding mantissas and then normalizing the result.

Machine Language and Instruction Formats categorize MIPS instructions into three types: R-type, I-type, and J-type. R-type instructions are for register-only operations (e.g., `add`, `sub`) and include fields for opcode, source registers (rs, rt), destination register (rd), shift amount (shamt), and function code (funct). I-type instructions are used for immediate arithmetic (e.g., `addi`) and load/store operations (e.g., `lw`, `sw`), containing an opcode, source/destination registers, and a 16-bit constant or address offset. Branch instructions (`beq`, `bne`) are also I-type and use PC-relative addressing. J-type instructions are for unconditional jumps (`j`, `jal`), with an opcode and a 26-bit address offset, which is combined with bits from the Program Counter (PC) to form the target address. Concepts like sign extension and zero extension are crucial for handling immediate values in I-type instructions.

In summary, a strong understanding of computer architecture, from the high-level organization to the intricacies of machine language and number representation, is essential for any computer science professional. The MIPS architecture serves as an excellent model for learning these core principles, providing practical insights into how hardware executes software and the fundamental operations that underpin all computing.