

MODULE 6: Instruction Set Architecture

Lecture 6.1

Instruction Formats

Prepared By:

- Scott F. Midkiff, PhD
- Luiz A. DaSilva, PhD
- Kendall E. Giles, PhD

Electrical and Computer Engineering
Virginia Tech

Lecture 6.1 Objectives

- Describe and identify different instruction formats related to the number and type of operands
- Describe key tradeoffs in instruction set design
- Define “big endian” and “little endian” schemes for storing multiple-byte data

Instructions

- Instructions need to indicate:
 - Operation to be performed (opcode)
 - Operands used as inputs and the output for the operation (or destination for jumps, etc.)
 - Various options, if any
- These can be represented as:
 - Mnemonics (assembly language instructions)
 - Binary codes (machine instructions)

Instruction Formats

- Different instruction set architectures (ISAs) support different instruction formats
- These different formats reflect:
 - Different design philosophies about how the ISA should support computation
 - Hardware and performance tradeoffs
 - Support for instruction set compatibility

What Differentiates Instruction Sets?

- The number of bits per instruction
 - 8, 16, 32, or 64 bits (or variable) are common
- The number of explicit operands allowed per instruction
 - 0, 1, 2, or 3 operands are typical
- Reference to operands
 - Operands may be in memory, general registers, an accumulator register, and/or in a stack
 - The instruction can specify different combinations of sources, e.g., register-to-register or memory-to-register
 - Operands may be of different lengths
- Types of operations and accessible operand locations

Design Factors Driving an ISA

- How much storage is required for instructions (still relevant for small form factor embedded processors)?
 - Number of bits per instruction
 - Number of instructions to represent a program
- How much processing (in hardware and time) is required to execute the instructions?
 - Complex instruction set computers (CISC)
 - Reduced instruction set computers (RISC)
- How effectively can compilers utilize the ISA?
 - Operation types for computation and control
 - Addressing modes

CISC versus RISC

CISC

- Complex instruction set
computers evolved as more functionality was added to processors
- Powerful instructions reduce program size
- Closely matched to high-level languages

More of a debate in the
1980's and 1990's

RISC

- Simple instructions with few operations and limited options
- Simplicity enables high clock rates (low execution time per instruction)
- Leverages compiler technology
- More “real estate” available for registers

CHECK POINT

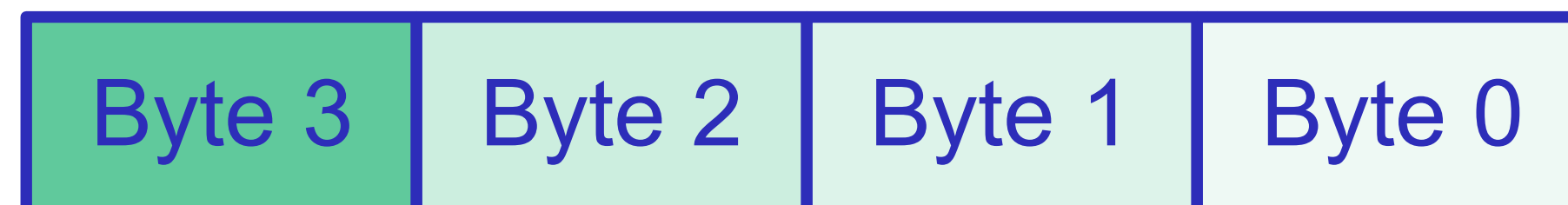
As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- Describe key tradeoffs in instruction set design

If you have any difficulties, please review the lecture video before continuing.

Byte Ordering for Storage

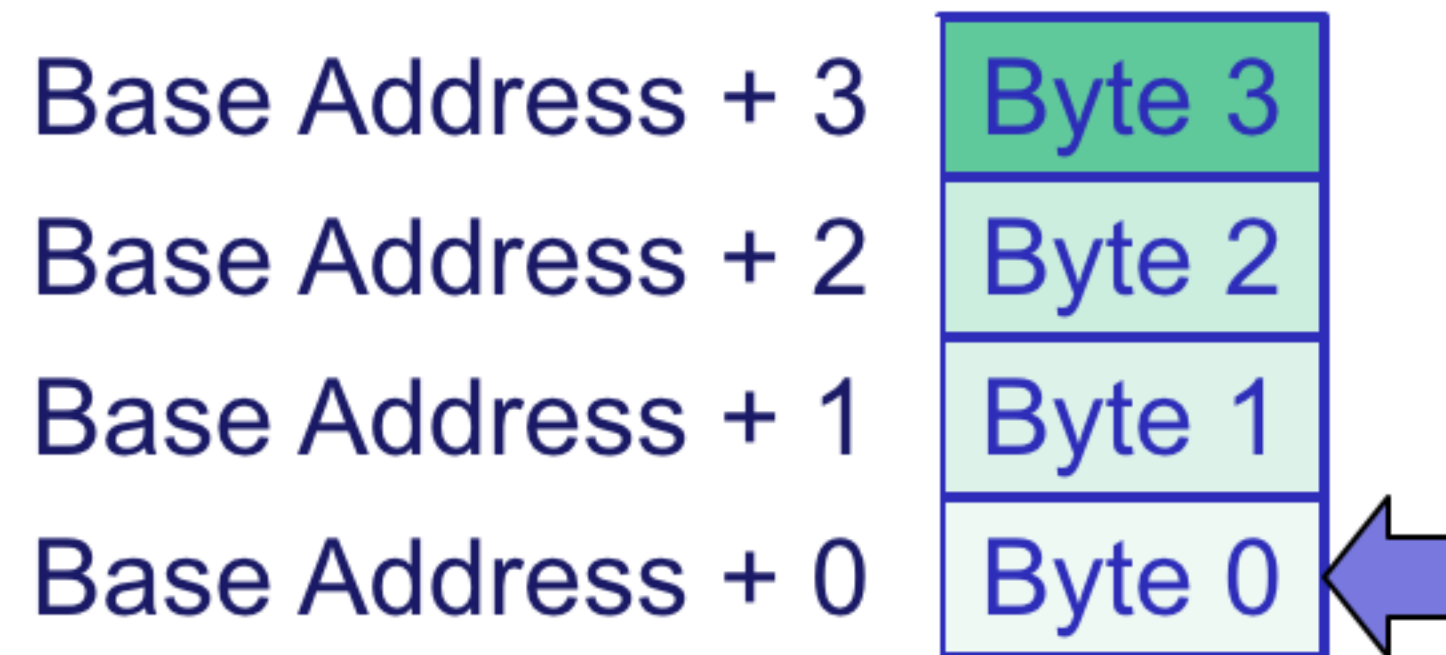
- The ISA must consider how multiple-byte values are stored
- Consider a four-byte value
 - An address points to a byte location
 - Which of the four bytes is this?
- Two approaches:
 - Little endian
 - Big endian



Little Endian versus Big Endian

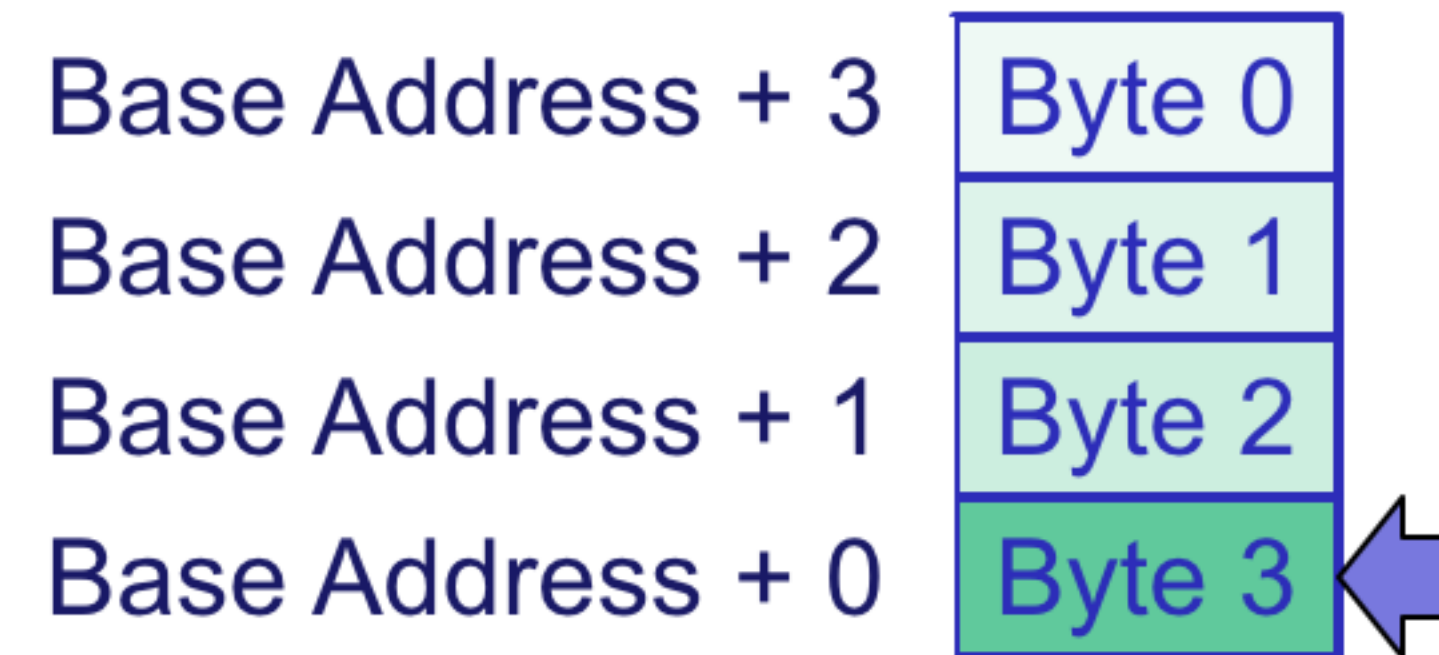
Little Endian

- Point to the least significant (the little) byte



Big Endian

- Point to the most significant (the big) byte




Little or Big Endian Example

Consider the hex value **12345678** stored at location 500

Little Endian

- Least significant byte (78) goes in location 500


503:	12
502:	34
501:	56
500:	78



Big Endian

- Most significant byte (12) goes in location 500

503:	78
502:	56
501:	34
500:	12



What if a value is first stored as little endian and then read as big endian?

Big Endian or Little Endian?

- There are advantages and disadvantages to both approaches
- The real disadvantage of either is that conversion is needed to the other since there is not universal agreement on which form to use
- Historical comments:
 - Intel's x86 architecture is little endian, while Sun and Motorola processors are big endian
 - Some processors support both
 - Network byte order in the Internet is big endian

Where are Operands Stored?

- Operands can be stored in the central processing unit (CPU) in:
 - A stack (stack architecture)
 - An accumulator register (an accumulator architecture)
 - General-purpose registers (a general-purpose register architecture)
- In practice, many CPUs combine two or more approaches, e.g., a processor may have both general-purpose registers and an accumulator

CHECK POINT

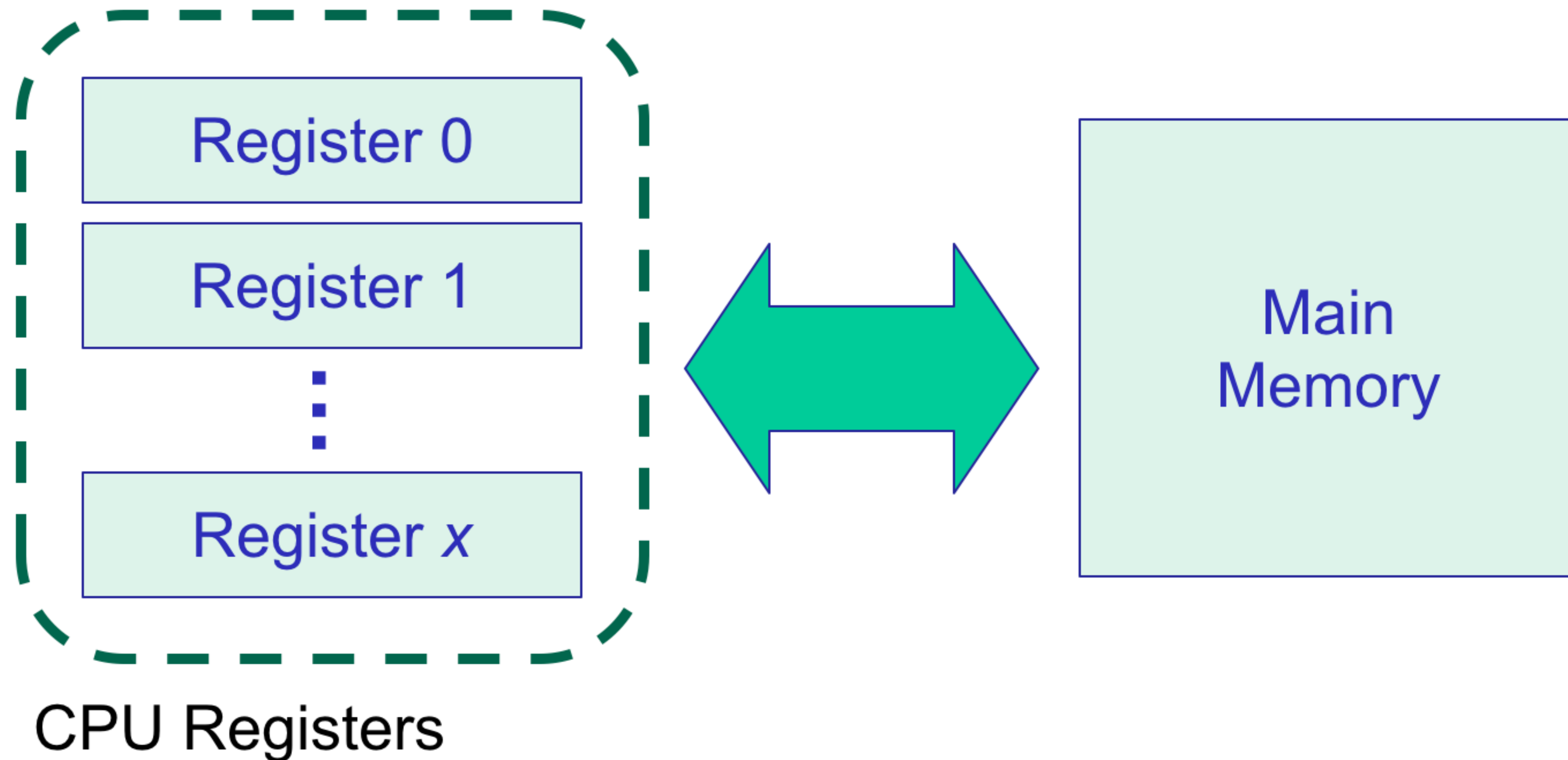
As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- Define “big endian” and “little endian” schemes for storing multiple-byte data

If you have any difficulties, please review the lecture video before continuing.

Accessing Memory

- The notion of general-purpose registers can be extended to include memory



Approaches to Accessing Memory

- Memory-memory architectures allow direct operation on memory locations without using a general-purpose register
- Register-memory architectures require a mix of operands in memory and at least one operand in a general-purpose register
- Load-store architectures require operands to be first moved from memory (with a “load”) and then stored to memory (with a “store”)
 - Could be called register-register architectures

Number of Operands

- Instructions can explicitly reference different numbers of operands
- Typical numbers of operands are:
 - 0 operands – operands are implied
 - 1 operand – just one operand is explicit while others, if present, are implicit
 - 2 operands – operands are explicit (one source is usually also the destination)
 - 3 operands – all operands are explicit (sources and destinations can all be different)

Instruction Operand Examples

ADD R1,R2,R3

Add the contents of register R1 to the contents of register R2 and store the result in register R3

ADD R1,R2

Add the contents of register R1 to the contents of register R2 and store in R1

ADD R1

Add the contents of R1 to the contents of the accumulator register (implied) and store result in the accumulator

ADD

Add the top two entries on the stack and push the result back on the stack

Instruction Length Tradeoffs

- More bits are needed to encode:
 - Larger number of operations
 - More operands
 - Larger number of possible operands
- But, simpler instructions may require that more instructions be executed
- Tradeoff in program storage and program execution time for different instruction sets is not so simple
 - Key issue in the RISC versus CISC debate
 - Involves hardware and clock speed factors

CHECK POINT

As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- Describe and identify different instruction formats related to the number and type of operands

If you have any difficulties, please review the lecture video before continuing.

Summary

- Different instruction set architectures (ISAs) have been defined based on design philosophies and tradeoffs
- Differentiated based on
 - Instruction length
 - The number and location of operands
 - How memory is accessed
- For multiple-byte data, “big endian” systems store the most significant byte in the base byte, while “little endian” systems store the least significant byte in the base byte

MODULE 6: Instruction Set Architecture

Lecture 6.1

Instruction Formats

Prepared By:

- Scott F. Midkiff, PhD
- Luiz A. DaSilva, PhD
- Kendall E. Giles, PhD

Electrical and Computer Engineering
Virginia Tech