MODULE 6: Instruction Set Architecture

Lecture 6.5
Real World Instruction Set Examples

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

Electrical and Computer Engineering
Virginia Tech

Lecture 6.5 Objectives

- Describe the basic features of the Intel x86 instruction set architecture
- Describe the (most) basic features of the MIPS instruction set architecture
- Describe key attributes of the Java Virtual Machine as an instruction set architecture

Instruction Set Architecture

- The instruction set architecture (ISA) is the view of the processor as seen by the assembly language programmer (or by the compiler for a high-level language)
- The ISA is a view of functional resources and capabilities, not of hardware and performance
- We consider three examples:
  - Part I – Intel x86 architecture
  - Part II – MIPS architecture
  - Part III – Java Virtual Machine (JVM) as an ISA

Part I – Intel x86 ISA Overview

- The Intel x86 architecture is a Complex Instruction Set Computer (CISC)
  - Instructions with variable length and execution time
  - Many operations can access data memory
  - Relatively complex functionality for some instructions
- 8086 and later processors support “real mode”
  - Segmented memory model
  - No supervisory mode (to protect operating system code)
- “Protected mode” introduced with 80286 and 80386
  - Flat memory model
  - Privileged instructions and protection

Intel x86 Registers

- Three sets of registers
  - General-purpose registers (mostly for data)
  - Address registers
  - Special-purpose registers (Flags, Instruction Pointer)
- 16-bit general-purpose registers, although some have special functions for some instructions
  - AX (AH is high byte, AL is low byte)
  - BX (BH is high byte, BL is low byte)
  - CX (CH is high byte, CL is low byte)
  - DX (DH is high byte, DL is low byte)

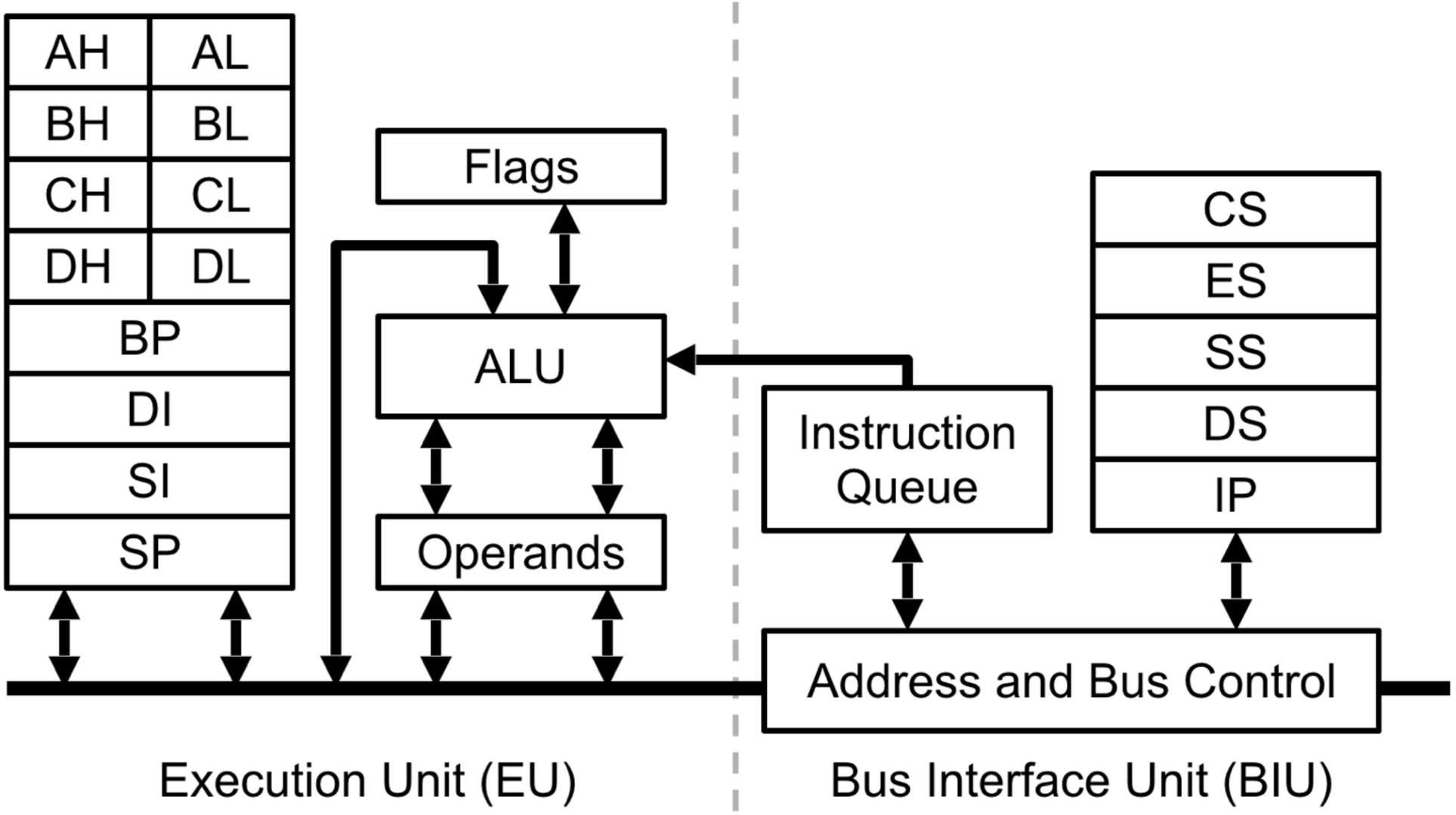
Intel x86 Registers (2)

- Address registers, e.g., for indexed or base addressing modes and for stack addressing
  - BP: Base Pointer
  - DI: Destination Index
  - SI: Source Index
  - SP: Stack Pointer
- Segment registers that point to segment in memory for code, data, and stack
  - CS: Code Segment register
  - DS: Data Segment register
  - ES: Extra (data) Segment register
  - SS: Stack Segment register

Intel x86 Registers (3)

- Flags register
  - Condition codes from ALU, e.g., sign flag (SF)
  - Exception bits, e.g., overflow flag (OF)
  - Other miscellaneous control functions, e.g. TRAP for single-step
- 16-bit Instruction Pointer (IP) register
  - Points to next instruction within the current code segment

Basic Datapath and Control — 8086



CHECK POINT

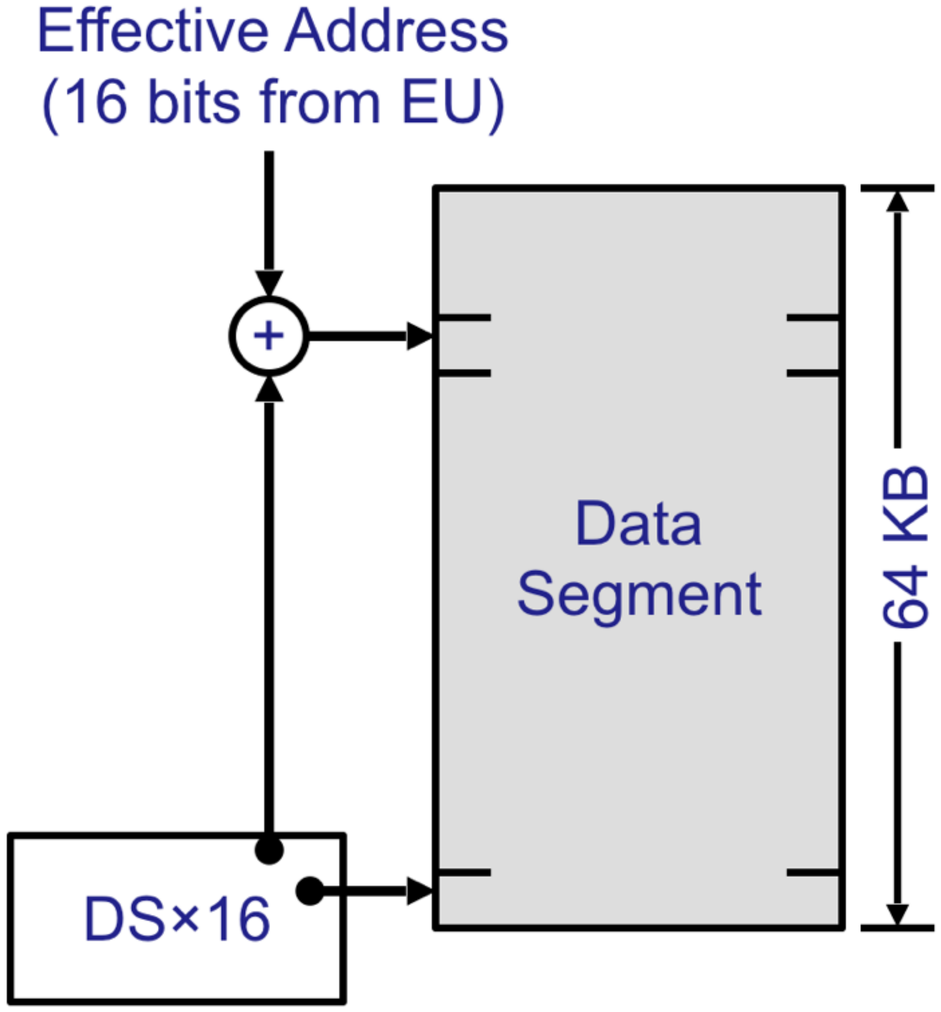
As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- Describe the registers of the x86 processor

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

Real Mode Segmented Memory

- Memory divided into segments of 64 kilobytes
- 16 bits can address all locations with a segment
- Segment register (shifted left four places) points to base of segment
- Effective address, e.g., for an operand, from the EU is added to the segment base to get memory address
- Special modes to access beyond segment boundary



Example Move Instructions

```
MOV  CL, AH       (move AH to CL – 8 bits)
MOV  CL, 30H      (move immediate data 30H to CL – 8 bits)
MOV  CX, AX       (move AX to CX – 16 bits)
MOV  CX, 0F34H    (move immediate data 0F34H to CX – 16 bits)
```

Example Complex Instruction: STOSB

- Instruction will copy a byte value into an arbitrary number of bytes beginning at an arbitrary address, e.g., to initialize a block of memory to all 0's
- Requires special use of registers
  - CX: Contains length of the block in bytes
  - AL: Contains value to be copied into bytes in block
  - DI: Contains address of beginning (or end) of block
- REP STOSB
  - Performs ([DI]) ← (AL); (DI) ← (DI) + 1; (CX) ← (CX) - 1
  - Repeats while (CX) > 0

Intel x86 Addressing Modes

- Register addressing
  - Operand is in a register
  - Instruction specifies register
- Immediate addressing
  - Instruction contains operand data
- Direct addressing
  - Operand is in memory
  - Instruction specifies memory location
  - Differences if reference is within current data segment or outside of current data segment

Intel x86 Addressing Modes (2)

- Register indirect addressing
  - Operand is in memory
  - Address of operand is contained in a register
  - Instruction specifies register that contains the operand address
- Based relative and index relative addressing
  - Like indirect, except a fixed offset is added to address in register to yield effective address
- Based indexed addressing
  - Like indexed, except address values from two registers plus a fixed offset are added to yield effective address

Real Versus Protected Modes

- Real-addressed mode
  - Original operating mode of x86 ISA beginning with 8086
  - Still supported on all x86 family processors
- Protected-addressed mode
  - Began with 80386 and carried forward
    - Limited features on 80286
  - Advanced on-chip features for larger systems and operating system support
    - Large, flat address space (64 terabytes)
    - Memory management
    - Multitasking
    - Protection

CHECK POINT

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

- Describe the x86 addressing modes and explain the difference between real and protected modes

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

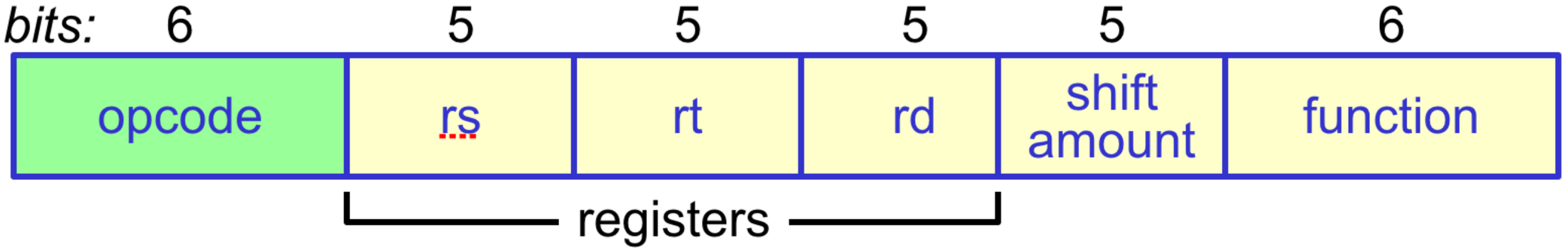
Part II – MIPS ISA Overview

- Developed by MIPS Technologies in 1985 and popular throughout the 1990s
- The MIPS architecture is a reduced instruction set computer (RISC) – perhaps *the* prototypical RISC processor
  - Fixed-length instructions
  - Execution time is fixed for a given type of instruction
  - Relatively simple functionality for each instruction
  - Load-store architecture, i.e., only the “load” and “store” instructions can access memory
  - 32 general-purpose registers

MIPS Instruction Formats

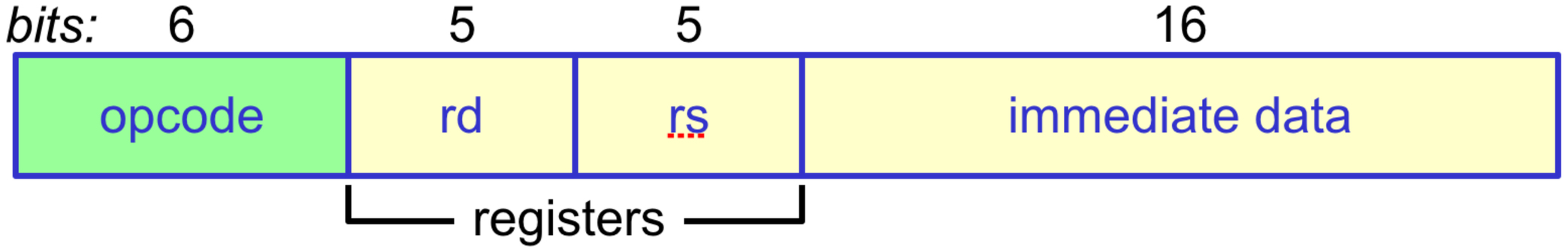
- Three formats
  - R (register)
  - I (immediate)
  - J (jump)
- All instructions are 32 bits long
- All instructions have a 6-bit opcode, with remaining 26 bits depending on type

MIPS R Type Instructions



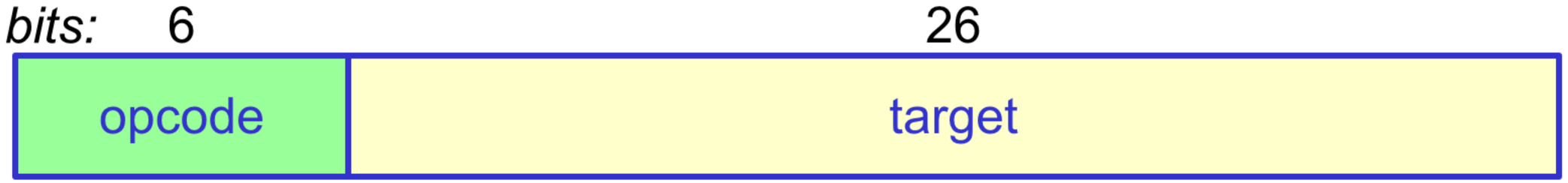
- R instructions reference registers
  - Three-operand instruction set
- Arithmetic and logical operations
- Typical form of an R type instruction
  - Assembly language: add $rd, $rs, $rt
  - Function: (rd) ← (rs) + (rt)

MIPS I Type Instructions



- I instructions reference registers and include an immediate operand
- Arithmetic operations, logic operations, and conditional jumps (with immediate data indicating the branch offset)
- Typical form of an I type instruction
  - Assembly language: add $rd, $rs, immediate
  - Function: (rd) ← (rs) + immediate

MIPS J Type Instructions



- J instructions are for jumps
- Typical form of a J type instruction
  - Assembly language: j target
  - Function: jump to location target

CHECK POINT

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

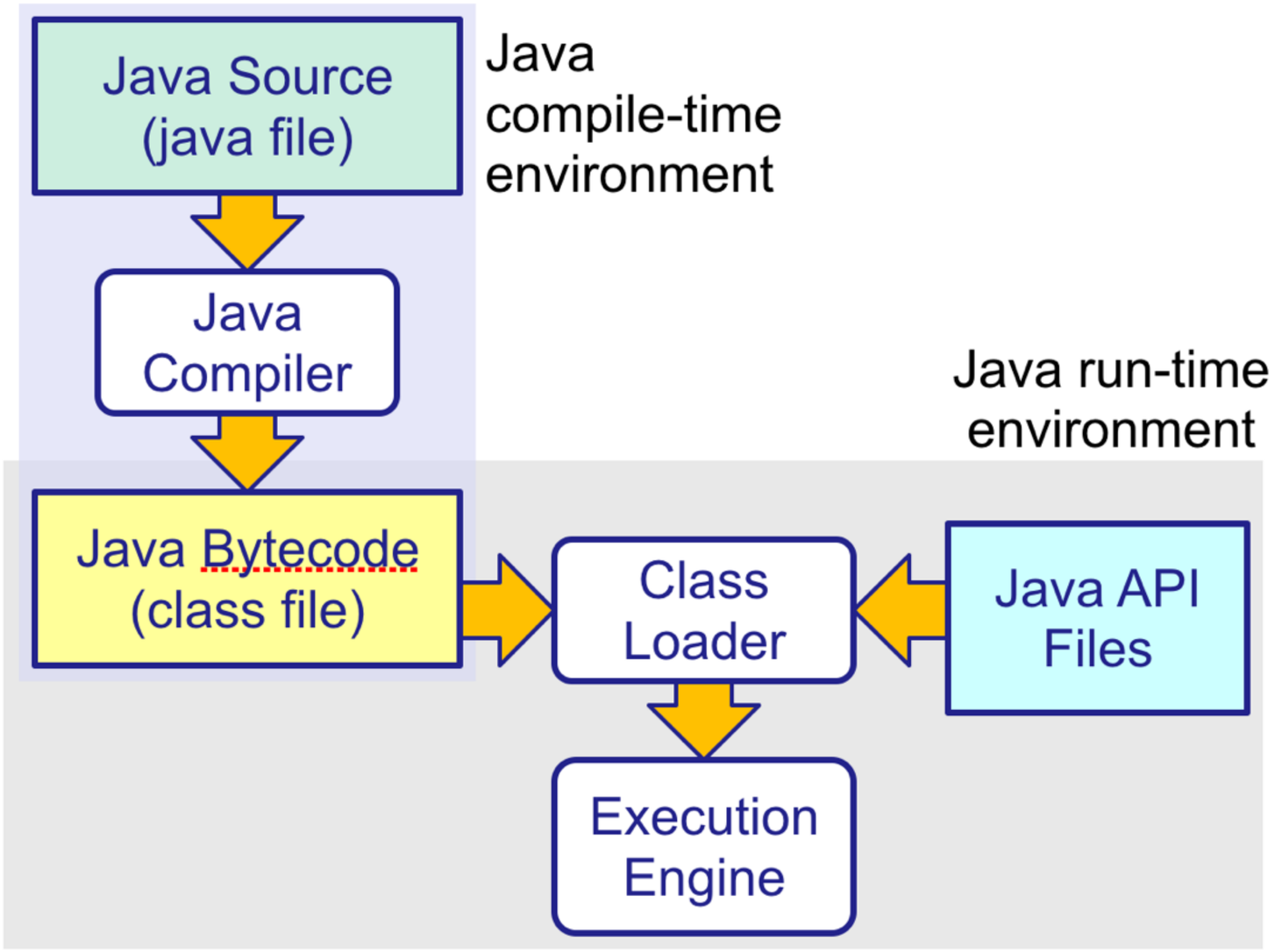
- Describe the basic features of the MIPS Instruction Set Architecture

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

Part III – Java Virtual Machine Overview

- The Java Virtual Machine (JVM) is “virtual” in that it is realized in software, which in turn executes in some other processor and operating system environment
  - Java source code is compiled into Java bytecode and then executed on the JVM
- Provides a common platform for development
  - Instruction set issues
  - Operating set issues
- Provides a protected environment
  - Protecting host system from Java program
  - Protected environment for Java program

Java Programming Environment



Java Bytecode

- Java Bytecodes are the assembly language (in mnemonic form) or machine language (in binary form) of the JVM
- All bytecodes are just that – one byte in length
    - Some instructions do require some additional following byte codes (e.g., for immediate data)
- Stack-based (0-operand) instructions

JVM Example: Java Source

```
Public class add {
  public static void main(String args[]) {
    int a=8, b=12, c=0;
    c=a+b;
  }
}
```

JVM Example: Java Bytecode

`bipush`	Push next byte on stack
`8`	Next byte is “8”
`istore_1`	Store as local variable 1 (a)
`bipush`	Push next byte on stack
`12`	Next byte is “12”
`istore_2`	Store as local variable 2 (b)
`iconst_0`	Push constant 0 on stack
`istore_3`	Store as local variable 3 (c)
`iload_1`	Push local variable 1 (a) on stack
`iload_2`	Push local variable 2 (b) on stack
`iadd`	Add top two stack elements and push
`istore_3`	Pop value from stack to local variable 3
`return`	Return from program

CHECK POINT

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

- Describe key attributes of the Java Virtual Machine as an instruction set architecture

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

Summary

- Intel x86 instruction set architecture – a CISC architecture
  - General-purpose data, address, and segment registers
  - Segmented memory model
  - Instruction examples
  - Addressing modes
- MIPS instruction set architecture – a RISC architecture
  - Fixed instruction lengths
  - R, I, and J instructions

Summary (2)

- Java Virtual Machine instruction set architecture
  - Java executes in a virtual machine – the JVM – that implements an instruction set architecture in software
  - The JVM runs on another processor, typically with a different ISA (although Java processors have been built)
  - Java source code is complied to bytecodes and bytecodes are executed by the JVM
  - The JVM is a stack-oriented ISA with 0-operand instructions

MODULE 6: Instruction Set Architecture

Lecture 6.5
Real World Instruction Set Examples

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

Electrical and Computer Engineering
Virginia Tech