MODULE 5: Assembly Language + Processor Control + Examples

Lecture 5.1
Assemblers

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

Electrical and Computer Engineering
Virginia Tech

Lecture 5.1 Objectives

- Identify the functions of assemblers, compilers, linkers, and loaders
- Describe the basic operation of an assembler
- Given a MARIE assembly language program, create a symbol table
- Compare and contrast assemblers and compilers

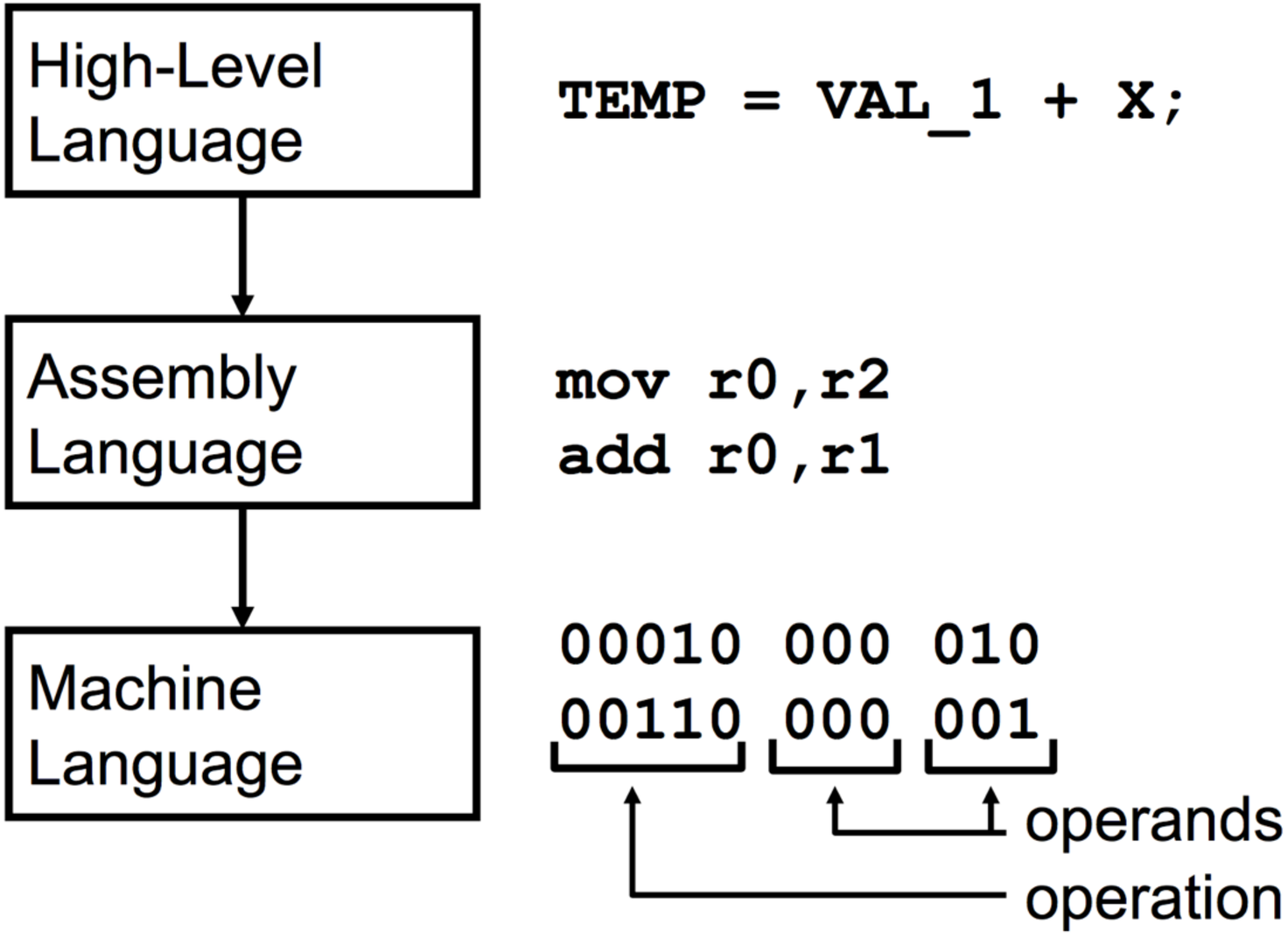
Metaprograms

- Metaprograms are programs that operate on other programs
- Metaprograms for running a high-level language program on an instruction set architecture (ISA)
    - Compiler
    - Assembler, possibly with macro capabilities
    - Linker
    - Loader

Language-Centric View of Computers (1)

- High-level language
  - Semantics of a problem or algorithm space
  - Machine and operating system independent
- Assembly language
  - Semantics of the computer, but in a human-readable form
  - Specific to a particular processor or processor family
- Machine language
  - Semantics of the computer in encoded, machine-readable form
  - A direct translation from assembly language

Language-Centric View of Computers (2)



Assemblers

- Assemblers translate instructions that are comprehensible to humans into the machine language that is comprehensible to computers
  - Alphanumeric instruction names (e.g., Load) instead of binary opcodes
  - Labels to identify or name particular memory addresses
- An assembler reads a source file (the assembly language program) and produces an object file (the machine code)

Assembler Operation

- Most assemblers perform their function in two “passes”
  - Pass 1 identifies all symbolic references to memory locations as well as the starting location of all instructions
  - Pass 2 converts all symbolic references to actual values and produces the final object code
- Symbol table: data structure that associates symbols and values
- Assembler must know opcode values, format, and length of all instructions

Example Revisited: Add Two Integers

- Add two integers (in memory locations 104 and 105) and store result (to memory location 106)
- Now using labels

Program

```
100        Load   X
101        Add    Y
102        Store  Z
103        Halt
104   X,   DEC 35
105   Y,   DEC -23
106   Z,   HEX 0000
```

Symbol Table
(created by the assembler)

X	104
Y	105
Z	106

Assembler Directives

- Instructions that are specific to the assembler and are not to be translated into machine code
- Examples
  - DEC, HEX instructs the assembler on how to interpret values
  - Comment delimiter (“/”) instructs the assembler to ignore all text following that character, until the end of the line

CHECK POINT

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

- Describe the function and basic operation of an assembler

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

Compilers

- A compiler is a metaprogram that converts a high-level language program into an assembly or machine language program for execution on a processor

Compilation

- Basic steps of compilation
  - Lexical analysis: identifying the basic symbols of the language
  - Parsing: identifying underlying program structure using symbols
  - Name analysis: associating names with variables and specific memory locations
  - Type analysis: identifying the type of data types
  - Action mapping and code generation: associating program statements with a sequence of assembly or machine language instructions

Compilers Versus Assemblers

- Compilers are complex
  - Map from the problem-oriented semantics of a high-level programming language to an instruction set architecture
  - Complex syntax of a high-level language
  - Opportunities for optimization
- Assemblers are relatively simple
  - Map between a symbolic representation of an ISA- specific program and a binary representation of the program
  - Simple syntax
  - No opportunities for optimization

Linking

- A “linker” combines separately assembled “object modules” into a single program
  - Creates a “load module”
- Functions of a linker
  - Resolve addresses that are external to a module
  - Relocate modules to merge them, e.g., in an end-to-end fashion
- Static versus dynamic linking
  - Traditional linkers link object modules prior to program execution
  - Dynamic link libraries allow components to be linked at run time

Loading

- Most user-level programs are compiled or assembled, so they may be dynamically loaded when ready for execution
  - Absolute memory addresses not known until loaded
  - Addresses are “relocatable” in memory, except for I/O addresses and other special locations
- Loader functions
  - “Locate” program module by mapping relative relocatable addresses to absolute addresses or initialize a base register for the module with an appropriate value
  - Load program module into memory
  - Initialize registers, such as stack pointer and program counter

CHECK POINT

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

- Identify the functions of compilers, linkers, and loaders
- Compare and contrast assemblers and compilers

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

Summary

- Examined four “metaprograms”
- Compiler
  - Complex operation
  - Analysis and code generation
  - Optimization
- Assembler
  - Two passes
  - Symbol table
- Linker
- Loader

Lecture 5.1
Assemblers

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

Electrical and Computer Engineering
Virginia Tech