computer science
illuminated

Low-Level Programming Languages

Nell Dale & John Lewis
(adaptation by Michael Goldwasser)

Computer Operations

- We have seen 0's and 1's used to represent all sorts of information (numbers, text, images, etc.)
- When it comes to programming a computer, the instructions for the computer are represented with 0's and 1's as well, and stored in main memory alongside the data. ("Stored Program Concept" – von Neumann architecture)

Machine Language

- **Machine language:** the instructions built into the hardware of a particular computer
- Every processor type has its own set of specific machine instructions
- The relationship between the processor and the instructions it can carry out is completely integrated
- Each machine-language instruction does only one very low-level task

Pep/7: A Virtual Computer

- A **virtual computer** is a hypothetical machine designed to contain the important features of real computers that we want to illustrated
- Pep/7 (designed by Stanley Warford)
  - has 32 machine-language instructions

    (so how many bits needed to represent?)
- We are only going to examine a few of these instructions

Features in Pep/7

- The memory unit is made up of 4,096 bytes of storage (so how many bits do we need to have distinct memory addresses?)
- Pep/7 has seven registers, four of which we focus on at this point
  - The program counter (PC) (contains the address of the next instruction to be executed)
  - The instruction register (IR) (contains a copy of the instruction being executed)
  - The index register (X register)
  - The accumulator (A register)

Features in Pep/7

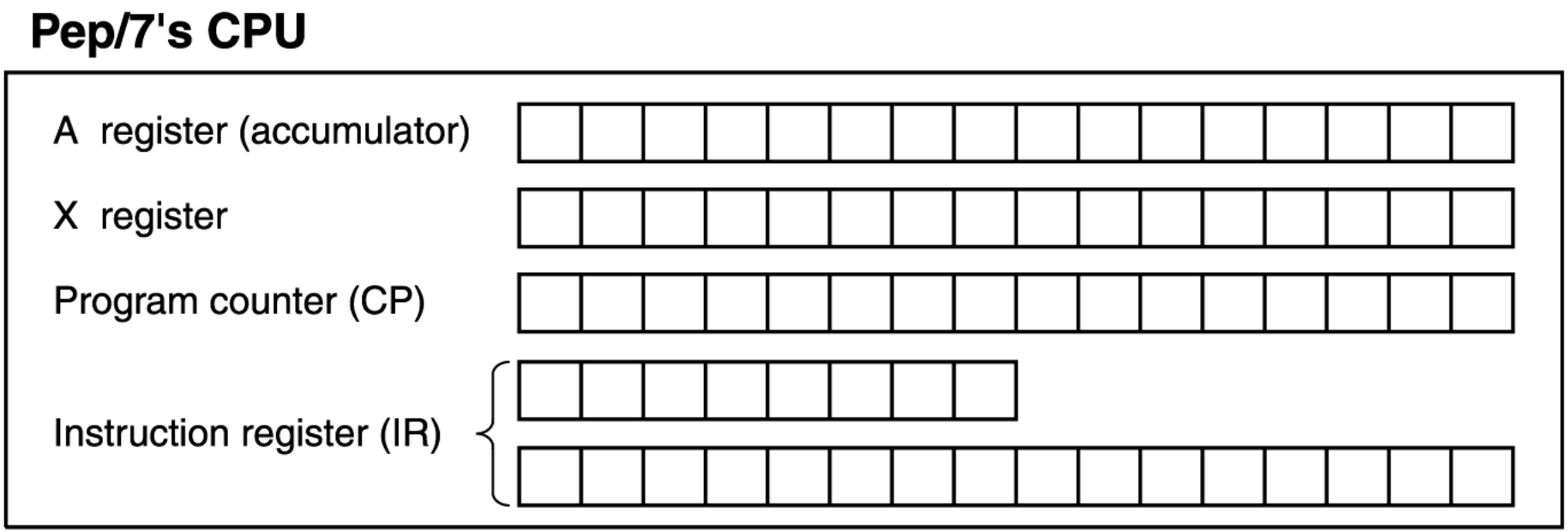


Figure 7.1 **Pep/7's architecture**

Instruction Format

- There are two parts to an instruction
  - The 8-bit instruction specifier
  - And optionally, the 16-bit operand specifier

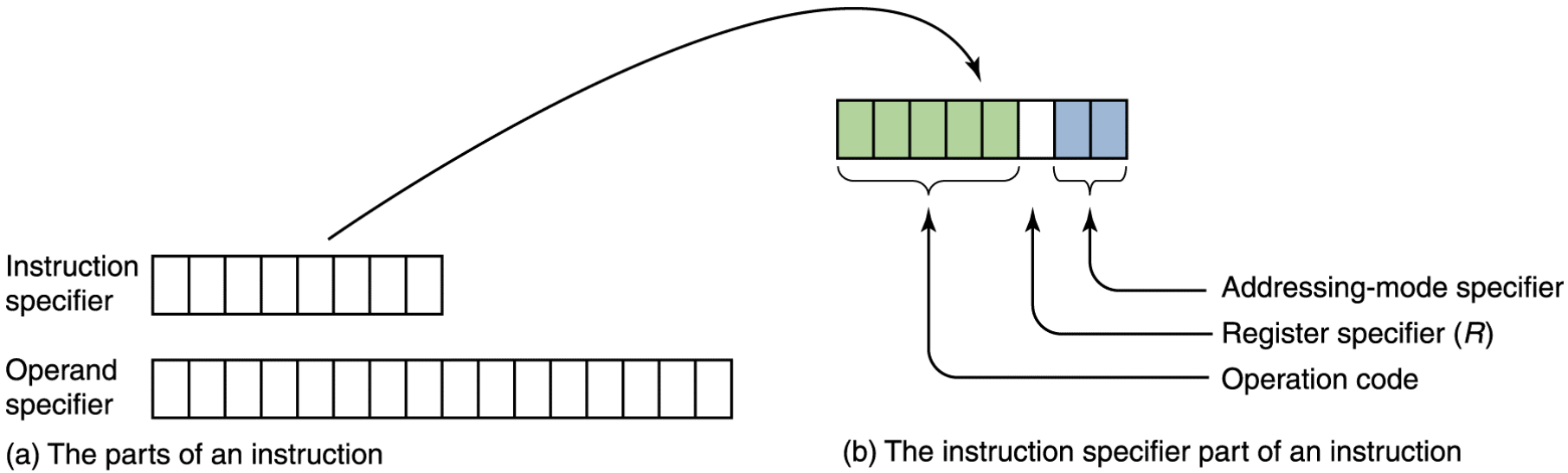


Figure 7.2 **The Pep/7 instruction format**

Instruction Format

- The instruction specifier is made up of several sections
  - The operation code
  - The register specifier
  - The addressing-mode specifier

Instruction Format

- The *operation code* specifies which instruction is to be carried out
- The 1-bit *register specifier* is 0 if register A (the accumulator) is involved in the operation and 1 if register X (the index register) is involved
- The 2-bit *addressing-mode specifier* says how to interpret the operand part of the instruction

Instruction Format

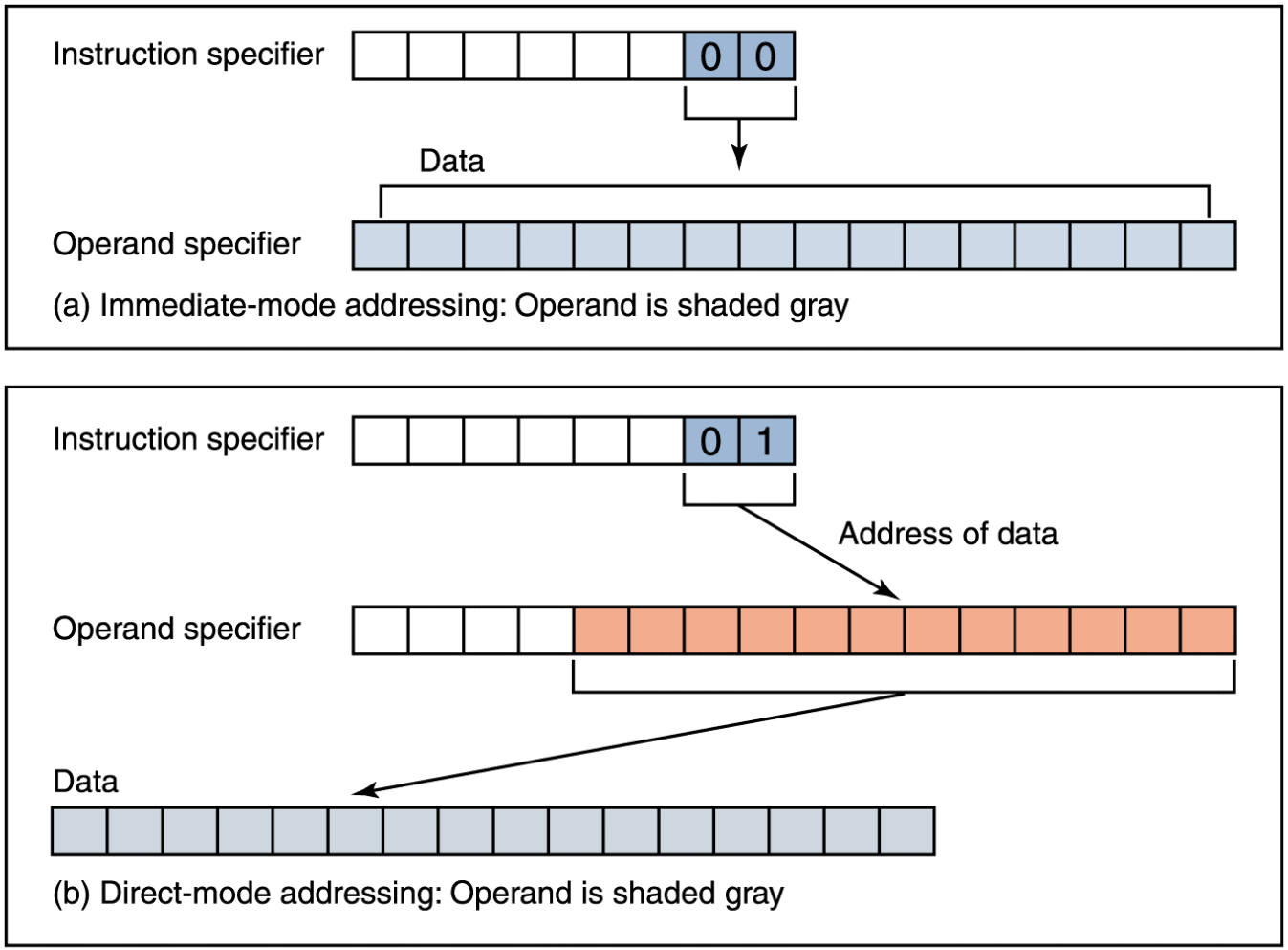


Figure 7.3 **Difference between immediate-mode and direct-mode addressing**

Some Sample Instructions

Opcode	Meaning of Instruction
00000	Stop execution
00001	Load operand into a register (either A of X)
00010	Store the contents of register (either A or X) into operand
00011	Add the operand to register (either A or X)
00100	Subtract the operand from register (either A or X)
11011	Character input to operand
11100	Character output from operand

Figure 7.3 **Subset of Pep/7 instructions**

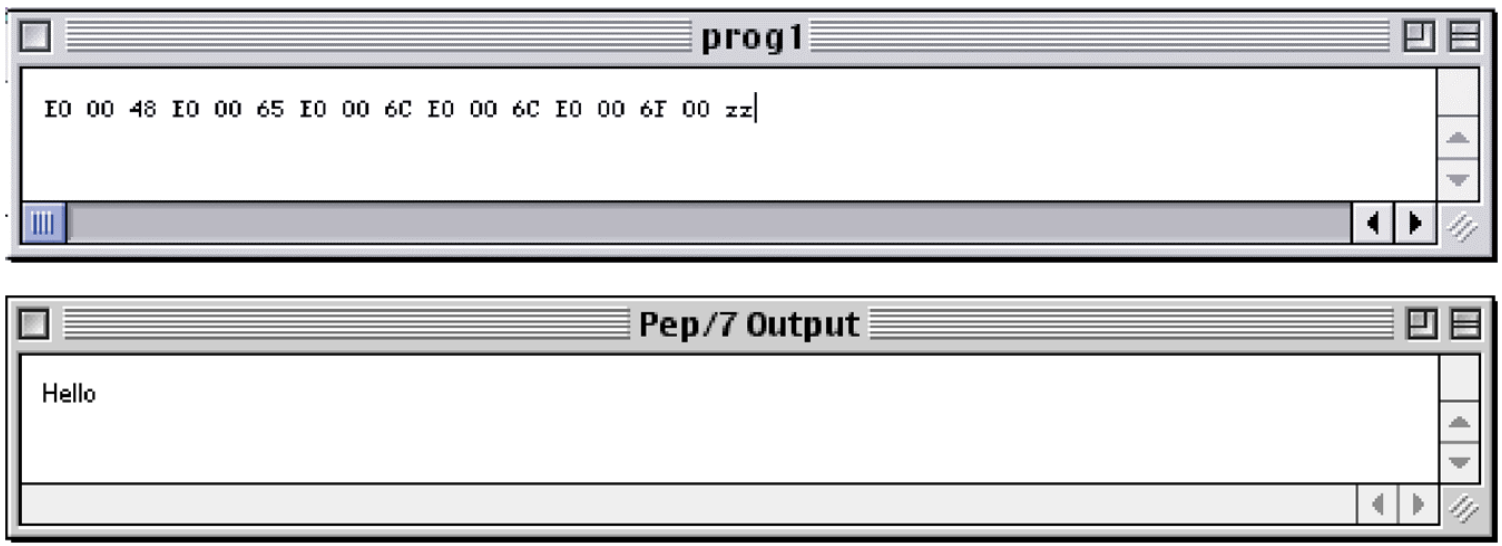
A Program Example

- Let's write "Hello" on the screen

Module	Binary Instruction	Hex Instruction
Write "H"	11100000 0000000001001000	E0 0048
Write "e"	11100000 0000000001100101	E0 0065
Write "l"	11100000 0000000001101100	E0 006C
Write "l"	11100000 0000000001101100	E0 006C
Write "o"	11100000 0000000001101111	E0 006F
Stop	00000000	00

Pep/7 Simulator

- A program that behaves just like the Pep/7 virtual machine behaves
- To run a program, we enter the hexadecimal code, byte by byte with blanks between each



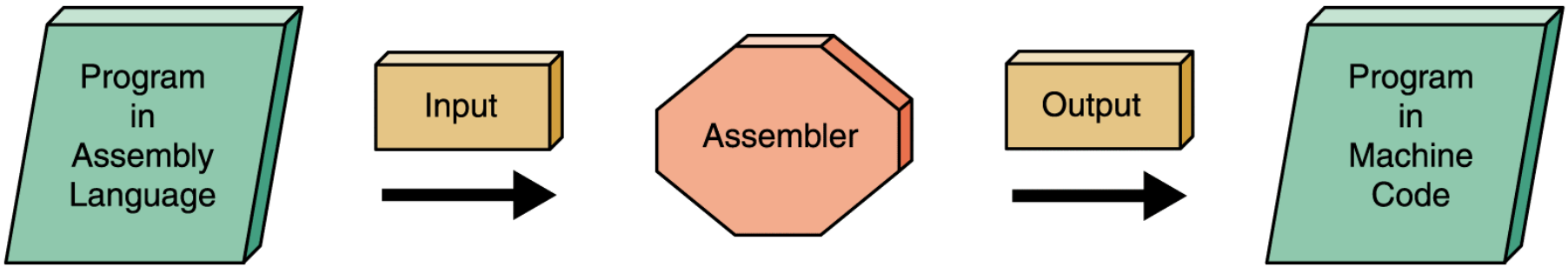
Assembly Language

- **Assembly languages:** assign mnemonic letter codes to each machine-language instruction
  - The programmer uses these letter codes in place of binary digits
  - A program called an assembler reads each of the instructions in mnemonic form and translates it into the machine-language equivalent

Pep/7 Assembly Language

Mnemonic	Operand, Mode Specifier	Meaning of Instruction
STOP		Stop execution
LOADA LOADA LOADX LOADX	h#008B,i h#008B,d h#008B,i h#008B,d	Load 008B into register A Load the contents of location 8B into register A Load 008B into register X Load the contents of 8B into register X
STOREA STOREX	h#008B,d h#008B,d	Store the contents of register A into location 8B Store the contents of register X into location 8B
ADDA ADDA ADDX ADDX	h#008B,i h#008B,d h#008B,i h#008B,d	Add 008B to register A Add the contents of location 8B to register A Add 008B to register X Add the contents of location 8B to register X
SUBA SUBA SUBX SUBX	h#008B,i h#008B,d h#008B,i h#008B,d	Subtract 008B from register A Subtract the contents of location 8B from register A Subtract 008B from register X Subtract the contents of location 8B from register X
CHARI	h#008B,d	Read a character and store it into byte 8B
CHARO	c#'B',i h#008B,d	Write the character B Write the character stored in byte 0B
DECI DECO DECO	h#008B,d h#008B,i h#008B,d	Read a decimal number and store it into location 8B Write the decimal number 139 (8B in hex) Write the decimal number stored in 8B

Figure 7.5 Assembly Process



A New Program

Set sum to 0
Read num1
Add num1 to sum
Read num2
Add num2 to sum
Read num3
Add num3 to sum
Write sum

Page 213

Our Completed Program

```
        BR Main         ;branch to location Main
sum:    .WORD d#0       ;set up word with zero as the contents
num1:   .BLOCK d#2      ;set up a two byte block for num1
num2:   .BLOCK d#2      ;set up a two byte block for num2
num3:   .BLOCK d#2      ;set up a two byte block for num3
Main:   LOADA sum,d     ;load a copy of sum into accumulator
        DECI num1,d     ;read and store a decimal number in num1
        ADDA num1,d     ;add the contents of num1 to accumulator
        DECI num2,d     ;read and store a decimal number in num2
        ADDA num2,d     ;add the contents of num2 to accumulator
        DECI num3,d     ;read and store a decimal number in num3
        ADDA num3,d     ;add the contents of num2 to accumulator
        STOREA sum,d    ;store contents of the accumulator into sum
        DECO sum,d      ;output the contents of sum
        STOP            ;stop the processing
        .END            ;end of the program
```

Testing

- **Test plan:** a document that specifies how many times and with what data the program must be run in order to thoroughly test the program
- A **code-coverage** approach designs test cases to ensure that each statement in the program is executed
- **Data-coverage testing** is another approach; it designs test cases to ensure that the limits of the allowable data are covered

Ethical Issues: Software Piracy, Copyrighting

- Research indicated that, globally, 11.5 billion dollars were lost in the year 2000 to pirated software
- Advocates of open-source code believe that a program's original source code should be in the public domain
- Respecting the copyrights of software, if it is not open code, is important from a number of perspectives