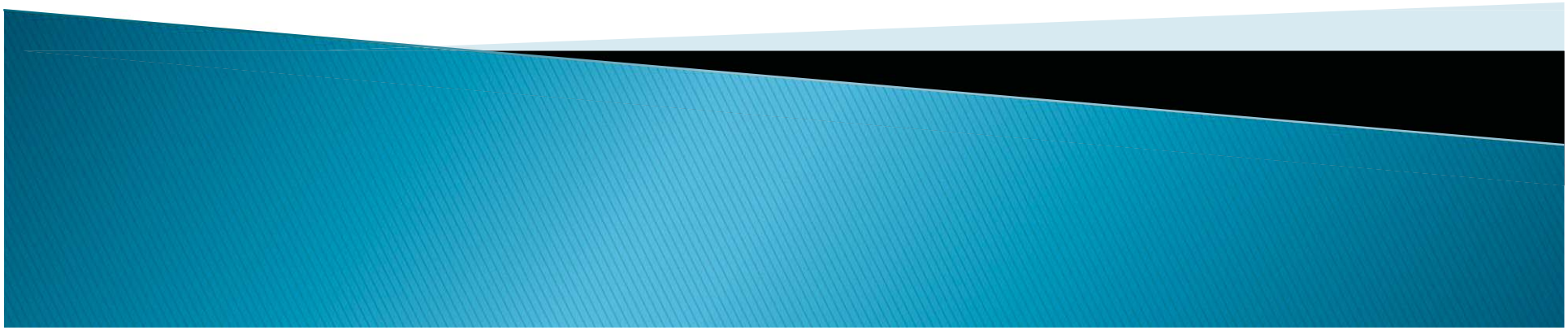
Low Level Programming

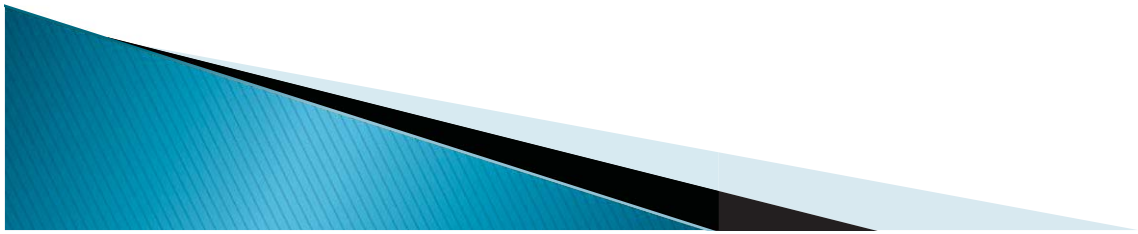
The way computers are programmed at machine level: Machine Language and Assembler

CHAPTER 6



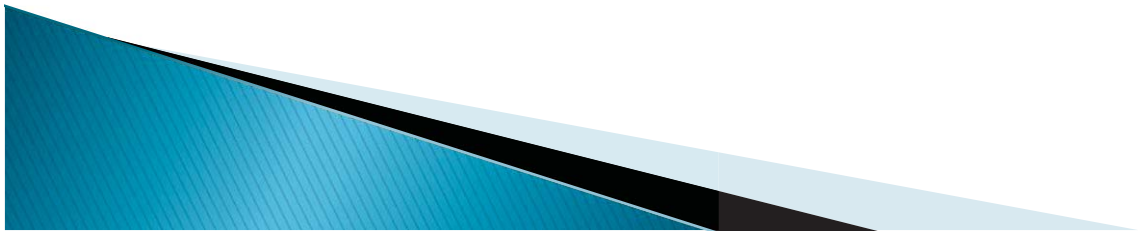
Chapter Goals

- Describe the important features of the Pep/8 virtual machine
- Distinguish between immediate mode addressing and direct addressing
- Convert a simple algorithm into a machine-language program
- Describe the Pep/8 simulator, and use it to run machine language programs



Chapter Goals

- Distinguish between machine language and assembly language
- Convert a simple algorithm into an assembly language program
- Distinguish between instructions to the assembler and instructions to be translated
- Use the Pep/8 simulator to assemble and run simple assembly language programs.

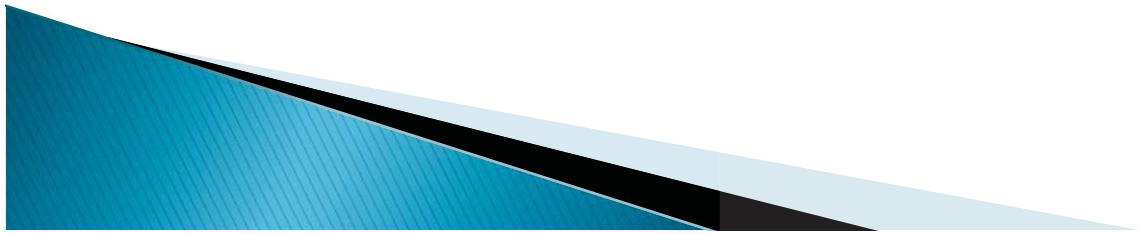


Computer Operations

Computer

A stored instruction electronic device that can store, retrieve, and process data

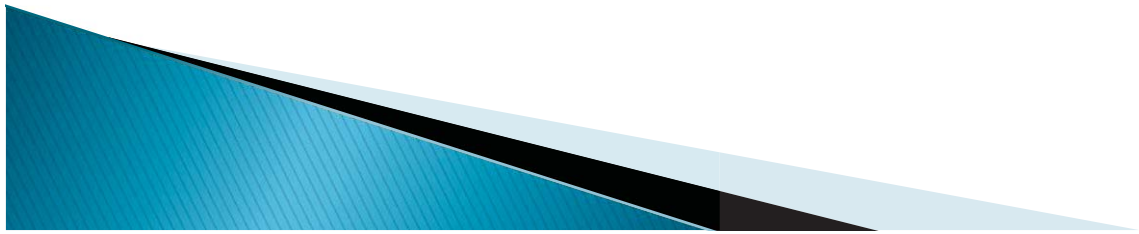
Data and **instructions to manipulate** the data are logically the same and can be stored in the same place



Machine Language

Machine language

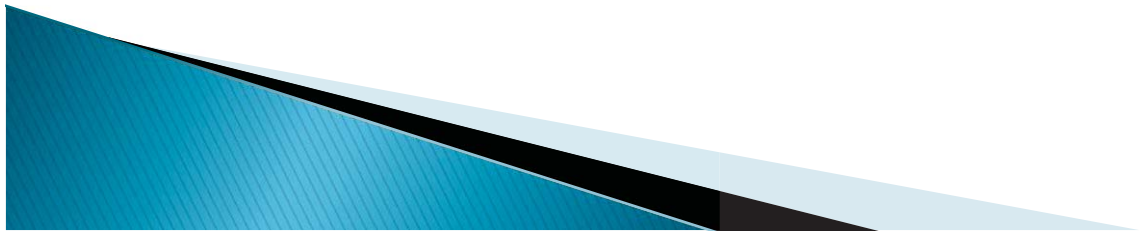
The language made up of binary coded instructions built into the hardware of a particular computer and used directly by the computer



Machine Language

Characteristics of machine language:

- 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



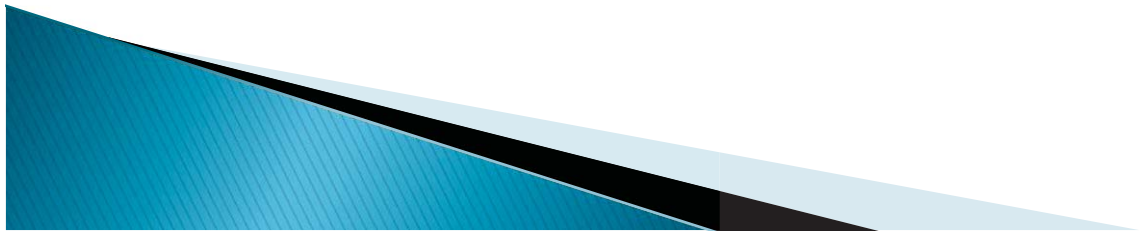
Pep/8: A Virtual Computer

Virtual computer

A **hypothetical** machine designed to demonstrate the important features of a real computer that we want to illustrate

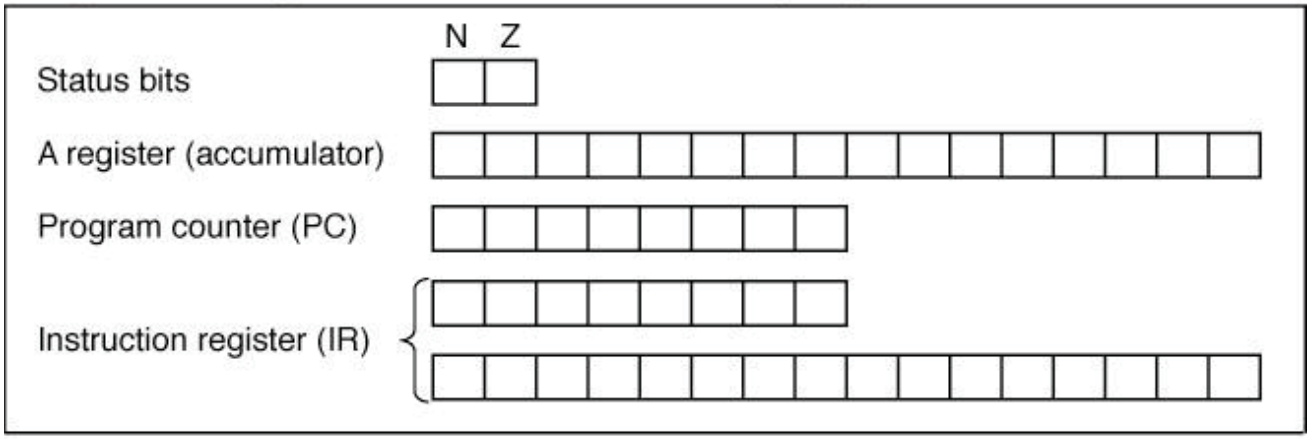
Pep/8

A virtual computer designed by Stanley Warford that has 39 machine-language instructions

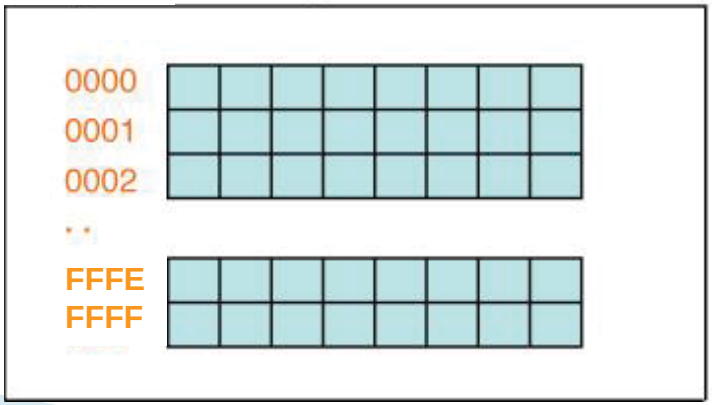


Features of Pep/8

Pep/8's CPU (as discussed in this chapter)



Pep/8's Memory



Features in Pep/8

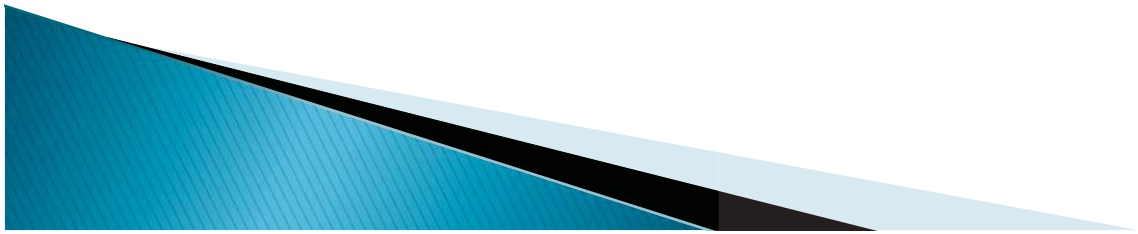
Pep/8 Registers & Status Bits

- 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 accumulator (A register)
- Status bit N (1 if register A is negative; 0 otherwise)
- Status bit Z (1 if the register A is 0; and 0 otherwise)

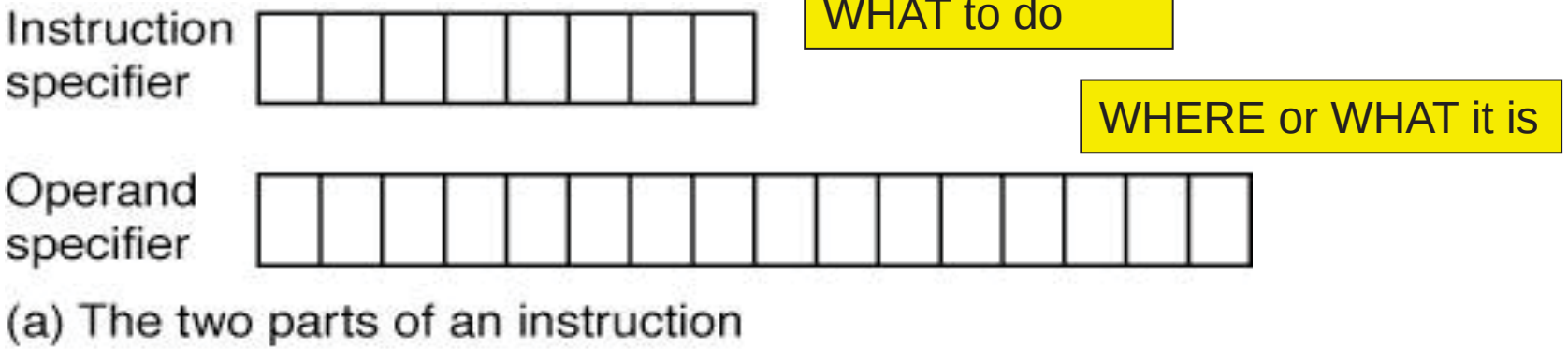
The memory unit is made up of 65 536 (16^4) bytes

What must an instruction do?

- Specify the OPERATION required e.g.
  - STOP the program
  - ADD values
  - STORE a value in memory
  - FIND something in memory
- Specify WHERE the action is to take place e.g.
  - Which register
- Specify WHERE the value is to be found or stored in memory…or specify the value itself.

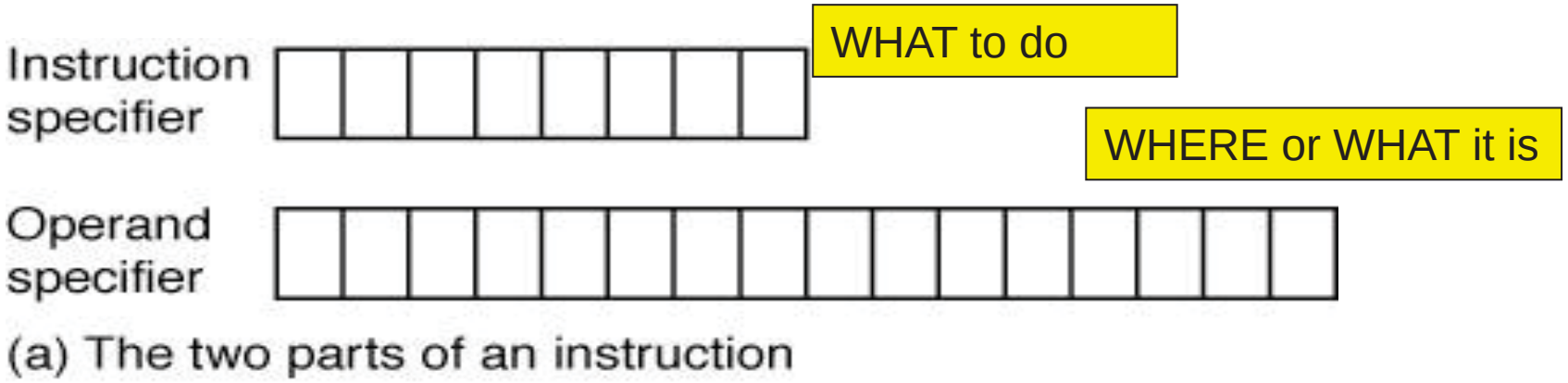


Instruction Format

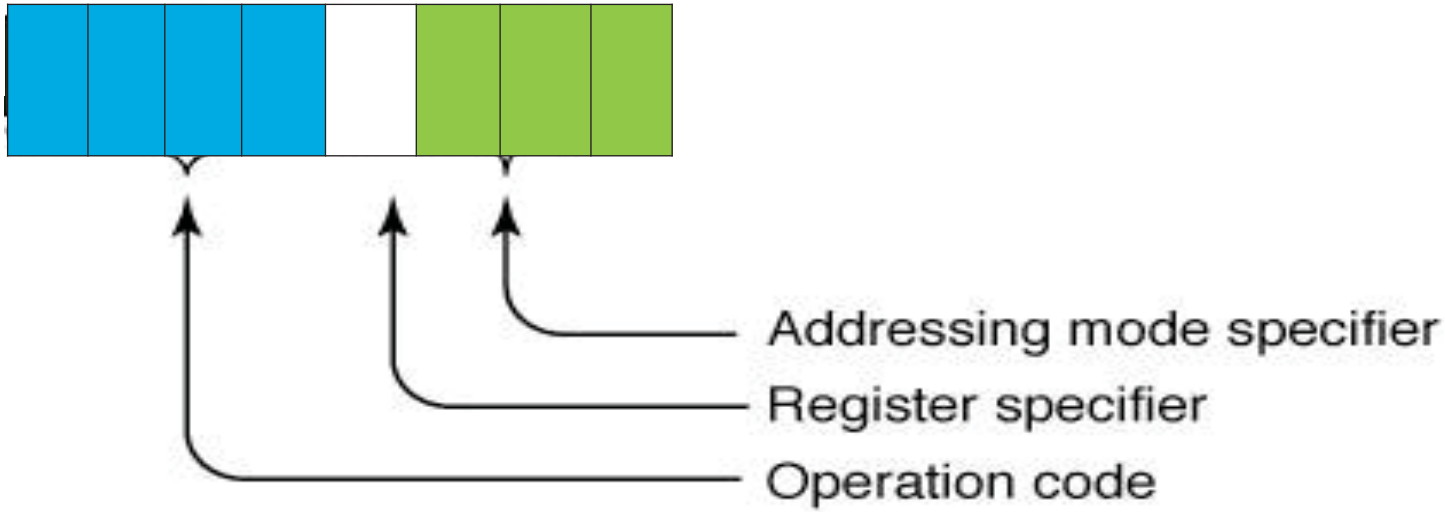


(a) The two parts of an instruction

Instruction Format



(a) The two parts of an instruction



(b) The instruction specifier part of an instruction

Instruction Format

Operation code

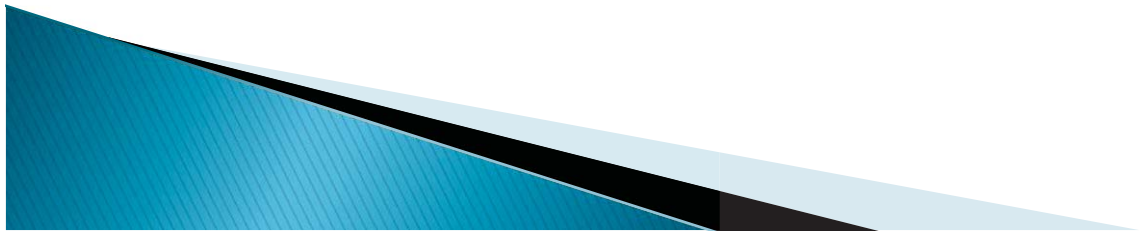
Specifies which instruction is to be carried out

Register specifier

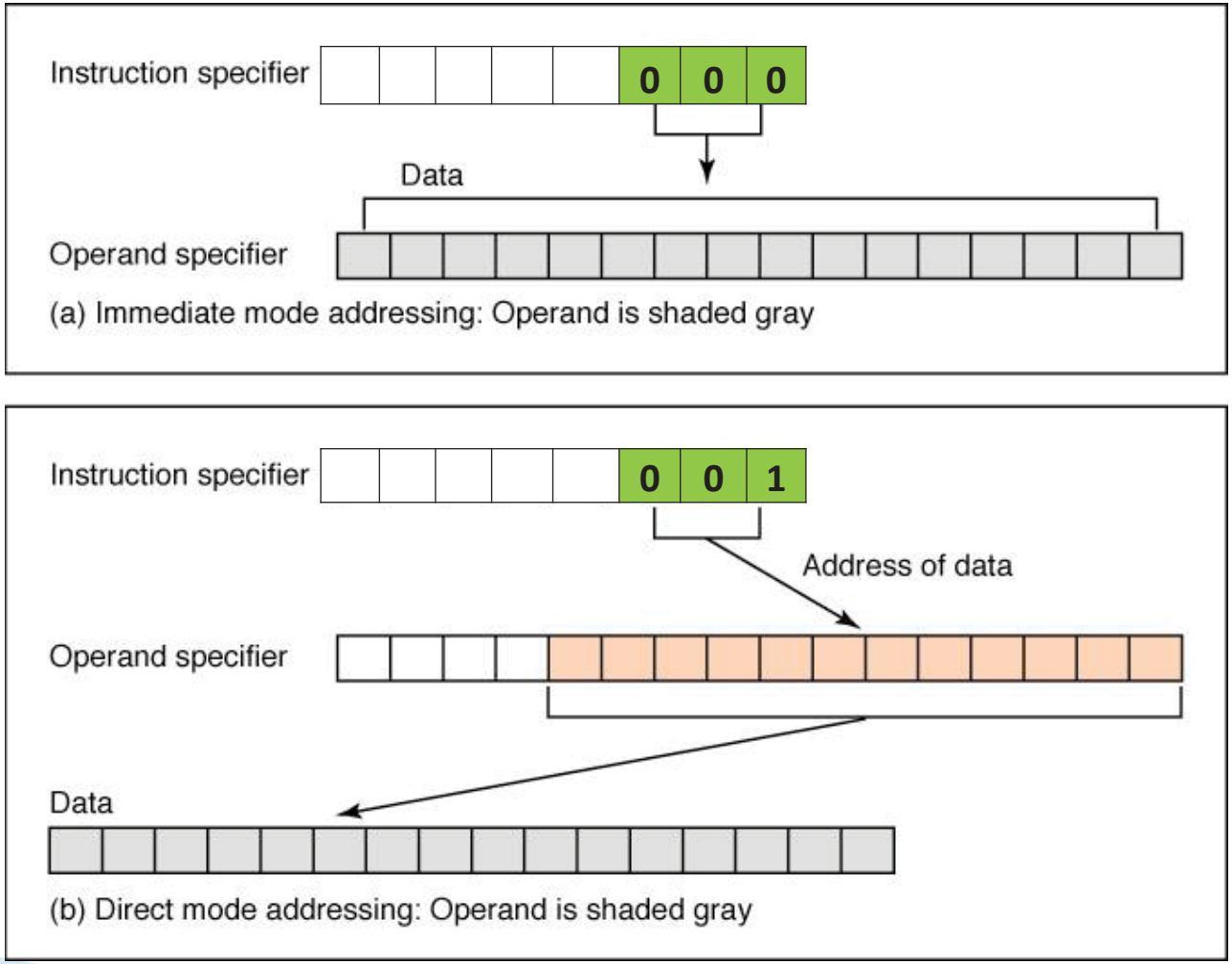
Specifies which register is to be used (we only use A)

Addressing-mode specifier

Says how to interpret the operand part of the instruction

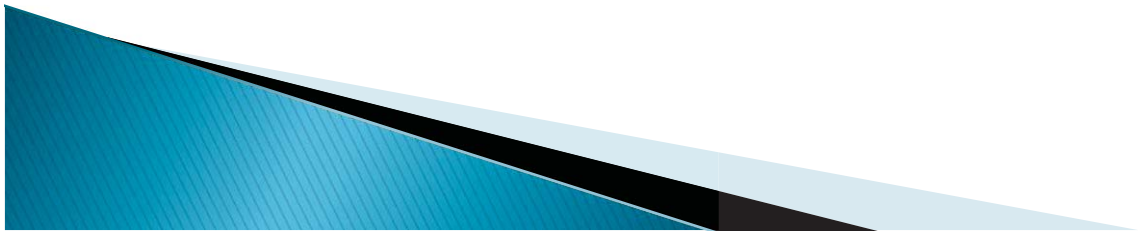


Instruction Format



Some Sample Instructions

Opcode	Meaning of Instruction
0000	Stop execution
1100	Load the operand into the A register
1110	Store the contents of the A register into operand
0111	Add the operand to the A register
1000	Subtract the operand from the A register
01001	Character input to the operand
01010	Character output from the operand



Some Sample Instructions

0000 STOP EXECUTION

1100 LOAD OPERAND into the A REGISTER

- Immediate addressing case:

  Instruction specifier:

  Operand specifier:

What happens?

- Direct addressing case:

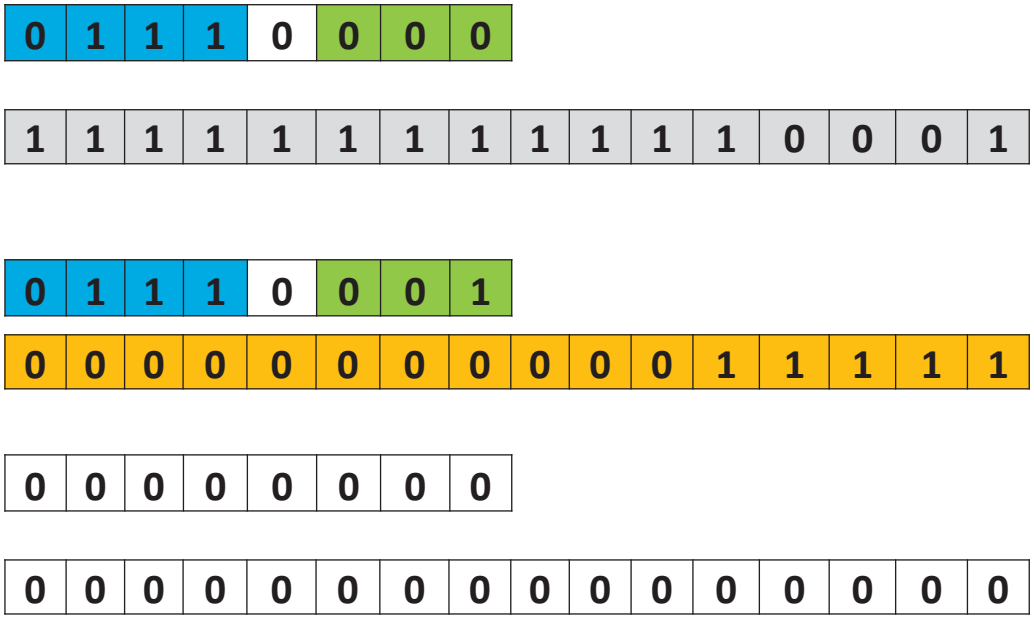
  Instruction specifier:

  Operand specifier:

What happens now?

What do these Instructions do?

Opcode	Meaning of Instruction
0000	Stop execution
1100	Load the operand into the A register
1110	Store the contents of the A register into operand
0111	Add the operand to the A register
1000	Subtract the operand from the A register
01001	Character input to the operand
01010	Character output from the operand



What do these Instructions do?

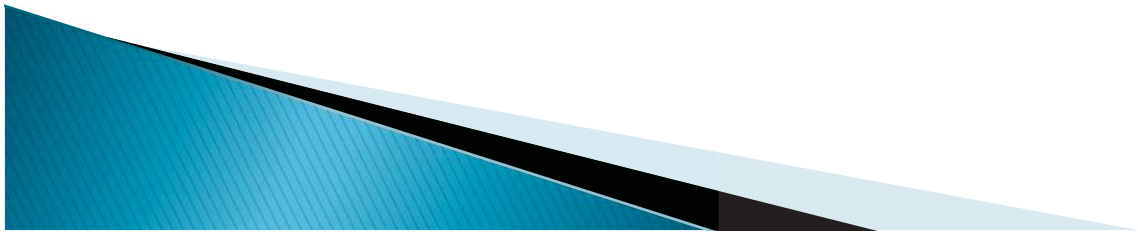
Opcode	Meaning of Instruction
0000	Stop execution
1100	Load the operand into the A register
1110	Store the contents of the A register into operand
0111	Add the operand to the A register
1000	Subtract the operand from the A register
01001	Character input to the operand
01010	Character output from the operand

1	1	1	0	0	0	0	1

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

1	1	1	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0



What do these Instructions do?

Opcode	Meaning of Instruction
0000	Stop execution
1100	Load the operand into the A register
1110	Store the contents of the A register into operand
0111	Add the operand to the A register
1000	Subtract the operand from the A register
01001	Character input to the operand
01010	Character output from the operand

0	1	0	0	1	0	0	1

0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0

0	1	0	1	0	0	0	0

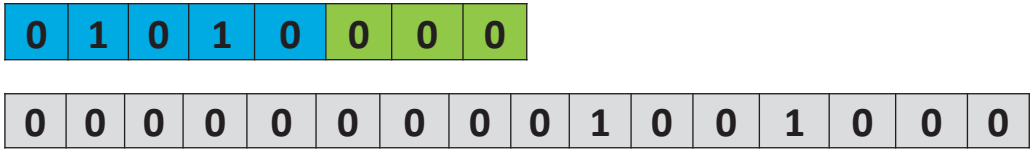
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1

0	1	0	0	1	0	0	1

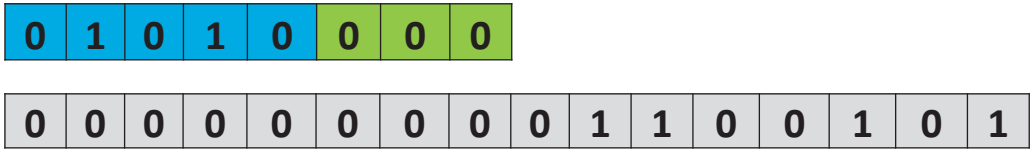
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1

A Program to ask for Help!

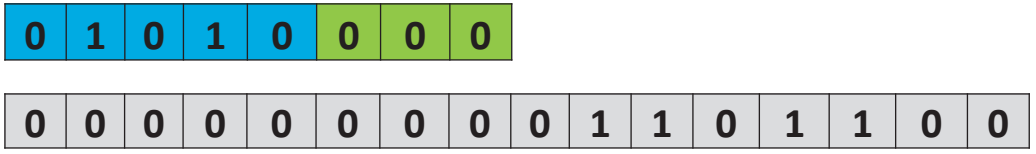
Opcode	Meaning of Instruction
0000	Stop execution
1100	Load the operand into the A register
1110	Store the contents of the A register into operand
0111	Add the operand to the A register
1000	Subtract the operand from the A register
01001	Character input to the operand
01010	Character output from the operand



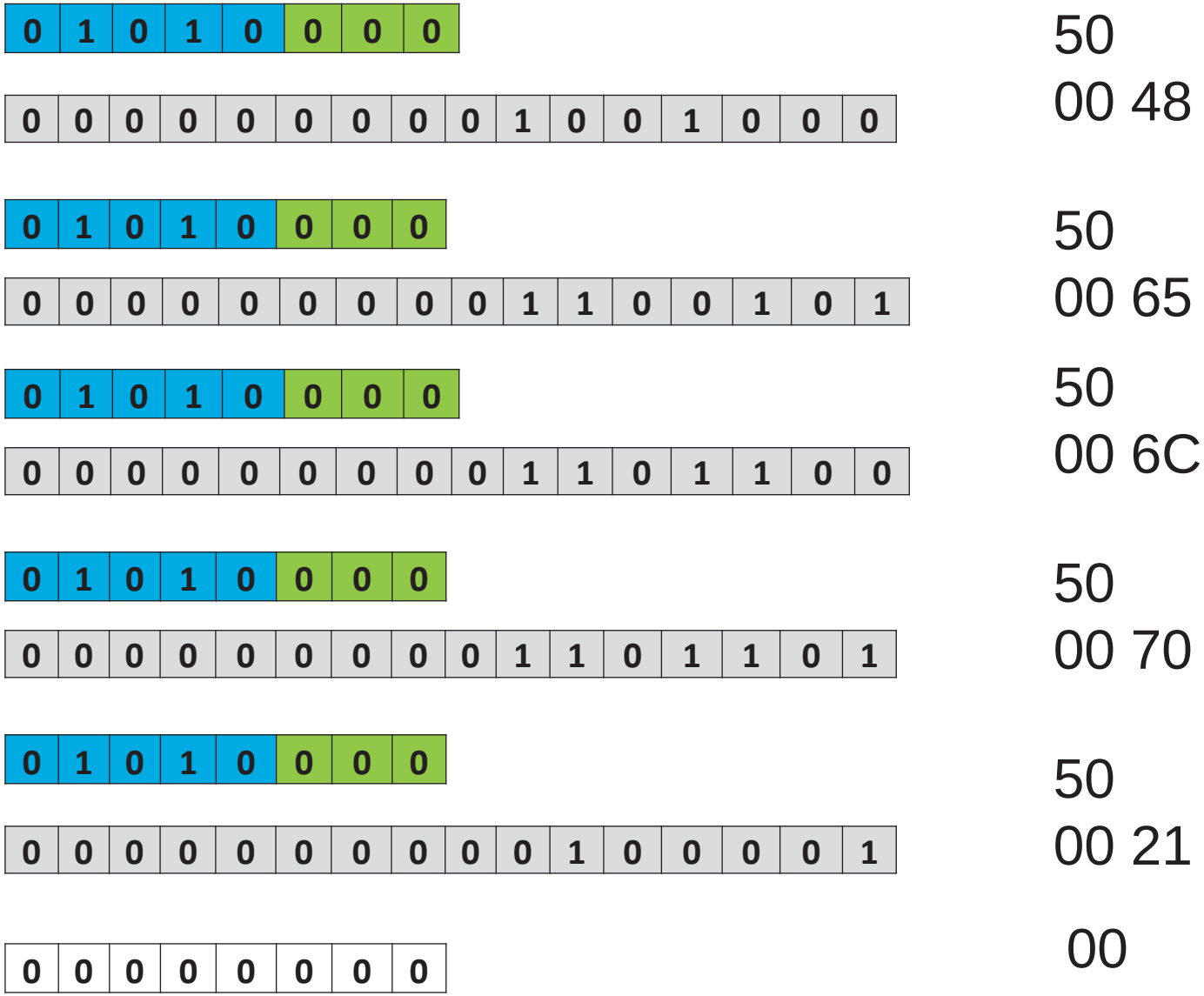
50
00 48



50
00 65



50
00 6C



50
00 48

50
00 65

50
00 6C

50
00 70

50
00 21

00

Pep/8 Simulator

Pep/8 Simulator

A program that behaves just like the Pep/8 virtual machine behaves

To run a program

Enter the hexadecimal code, byte by byte with blanks between each

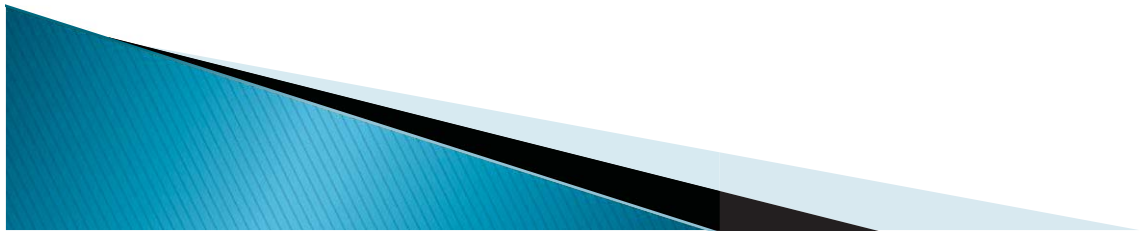
Terminate by inserting zz

Load the program

Run Object Code

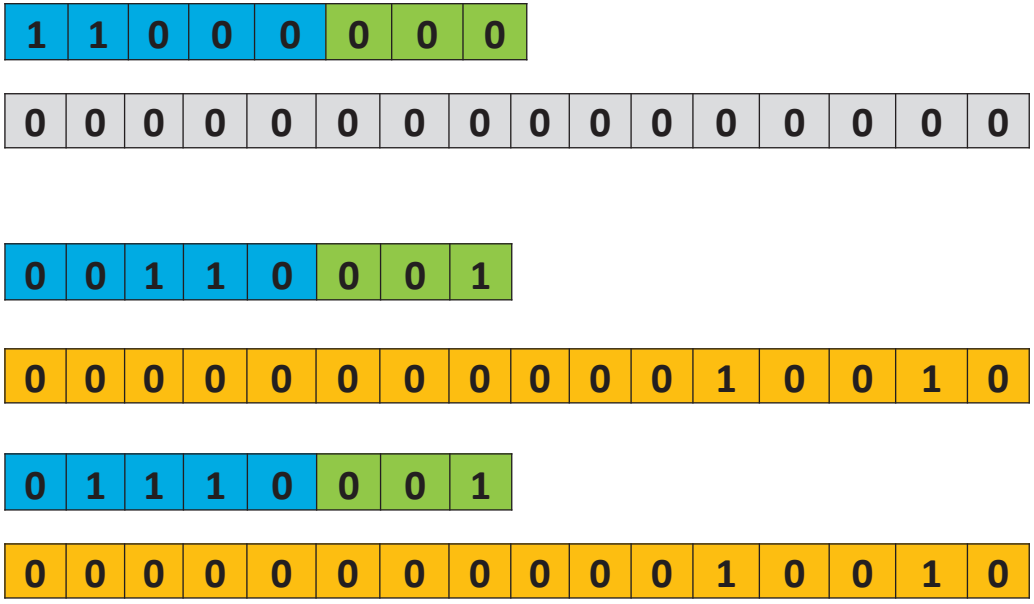
Pep/8 Simulator

- Download the Pep/8 Simulator from:
- http://code.google.com/p/pep8-1/
- Pep813Win.zip
- Now loaded on lab machines.



Program to Add Numbers

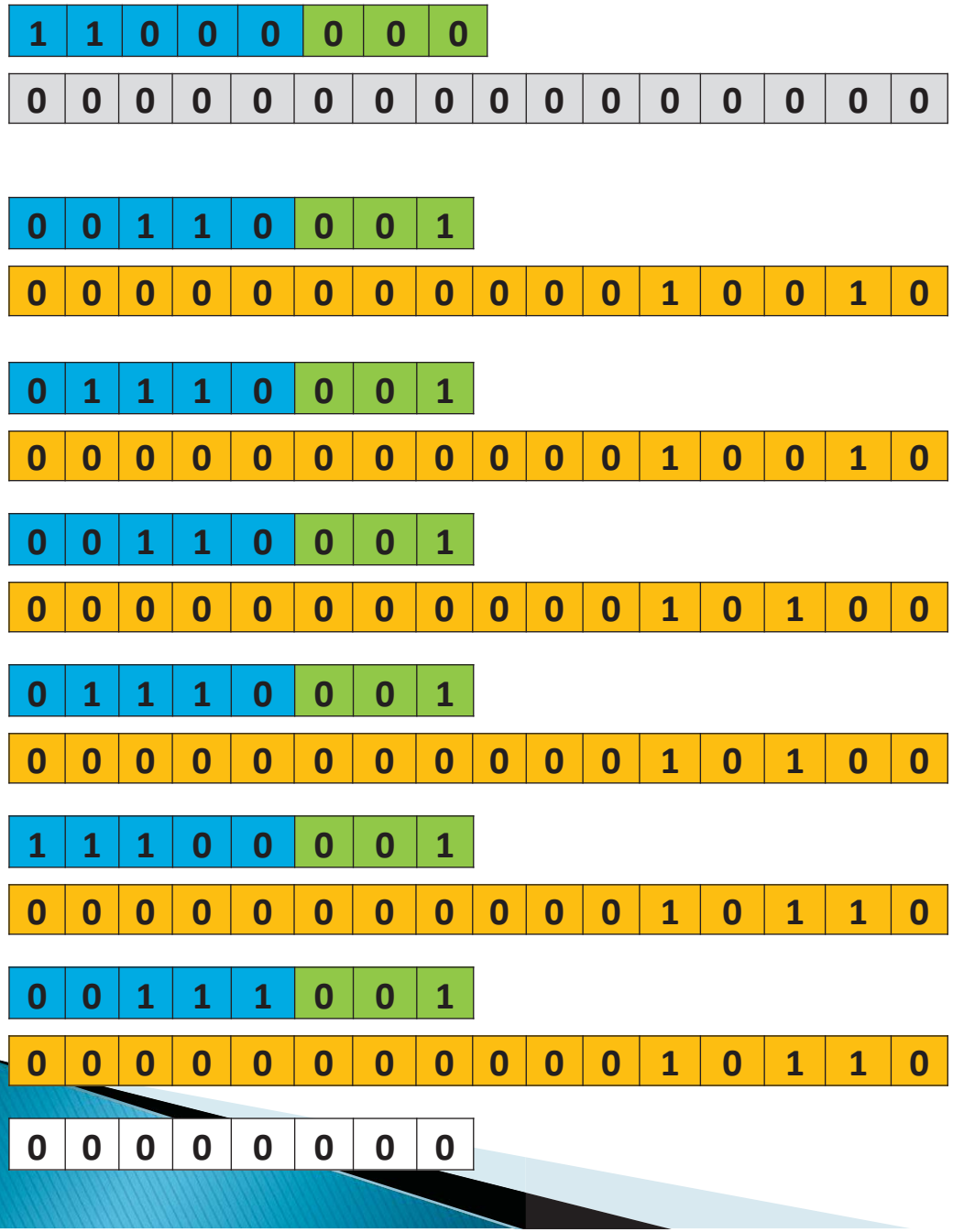
Opcode	Meaning of Instruction
0000	Stop execution
1100	Load the operand into the A register
1110	Store the contents of the A register into operand
0111	Add the operand to the A register
1000	Subtract the operand from the A register
00110	**Read in a decimal number**
00111	**Read out a decimal number**



C0
00 00

31
00 12

71
00 12



C0
00 00

31
00 12

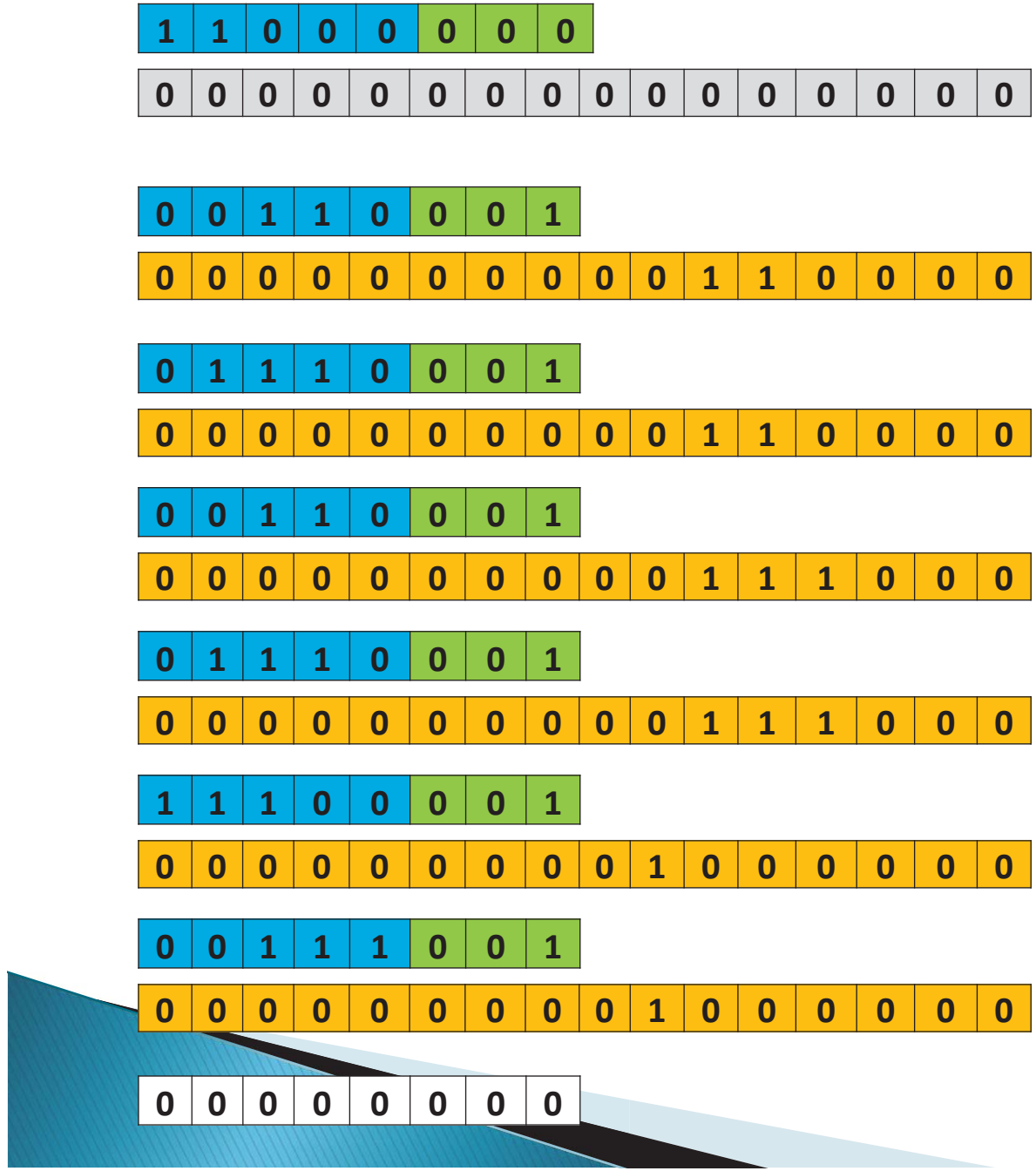
71
00 12

31
00 14

71
00 14

E1
00 16

39
00 16



C0
00 00

31
00 30

71
00 30

31
00 38

71
00 38

E1
00 40

39
00 40