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

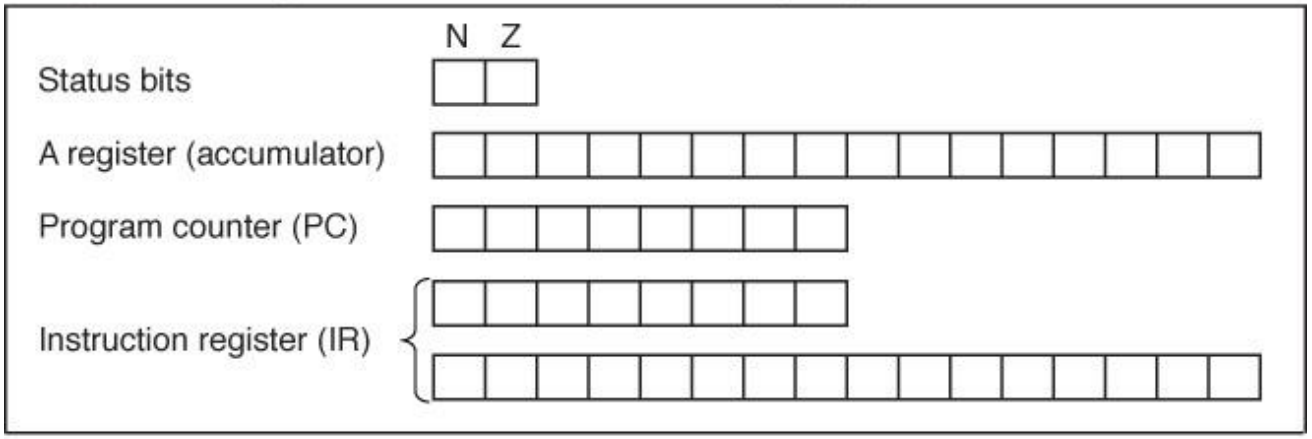
A **hypothetical** machine designed to demonstrate the important features of a real computer that we want to illustrate

Pep/8

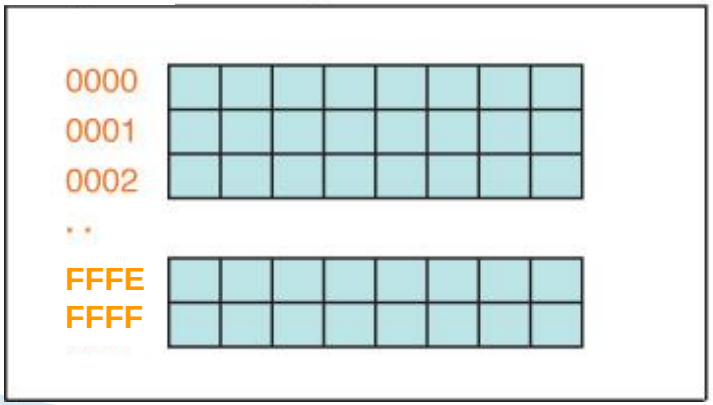
A virtual computer designed by Stanley Warford that has 39 machine-language instructions

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

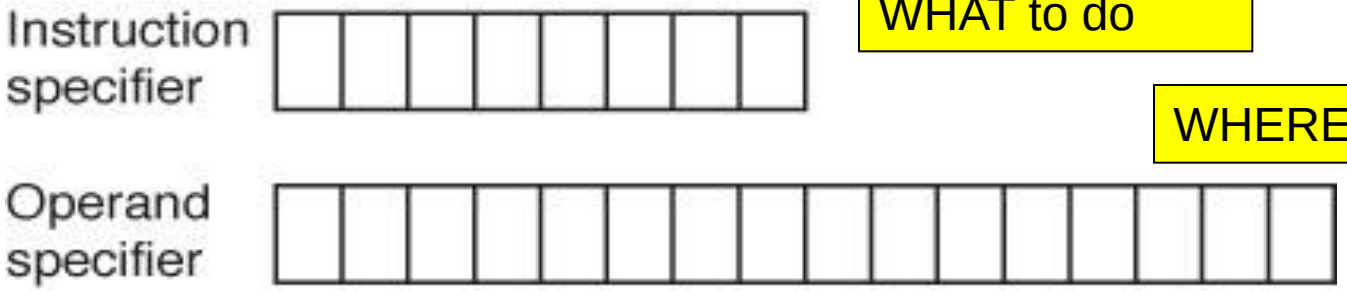
- The program counter (PC) (contains the address of the next instruction to be executed)
- The instruction register (IR) (contains a copy of the instruction being executed)
- The accumulator (A register)
- Status bit N (1 if register A is negative; 0 otherwise)
- Status bit Z (1 if the register A is 0; and 0 otherwise)

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

What must an instruction do?

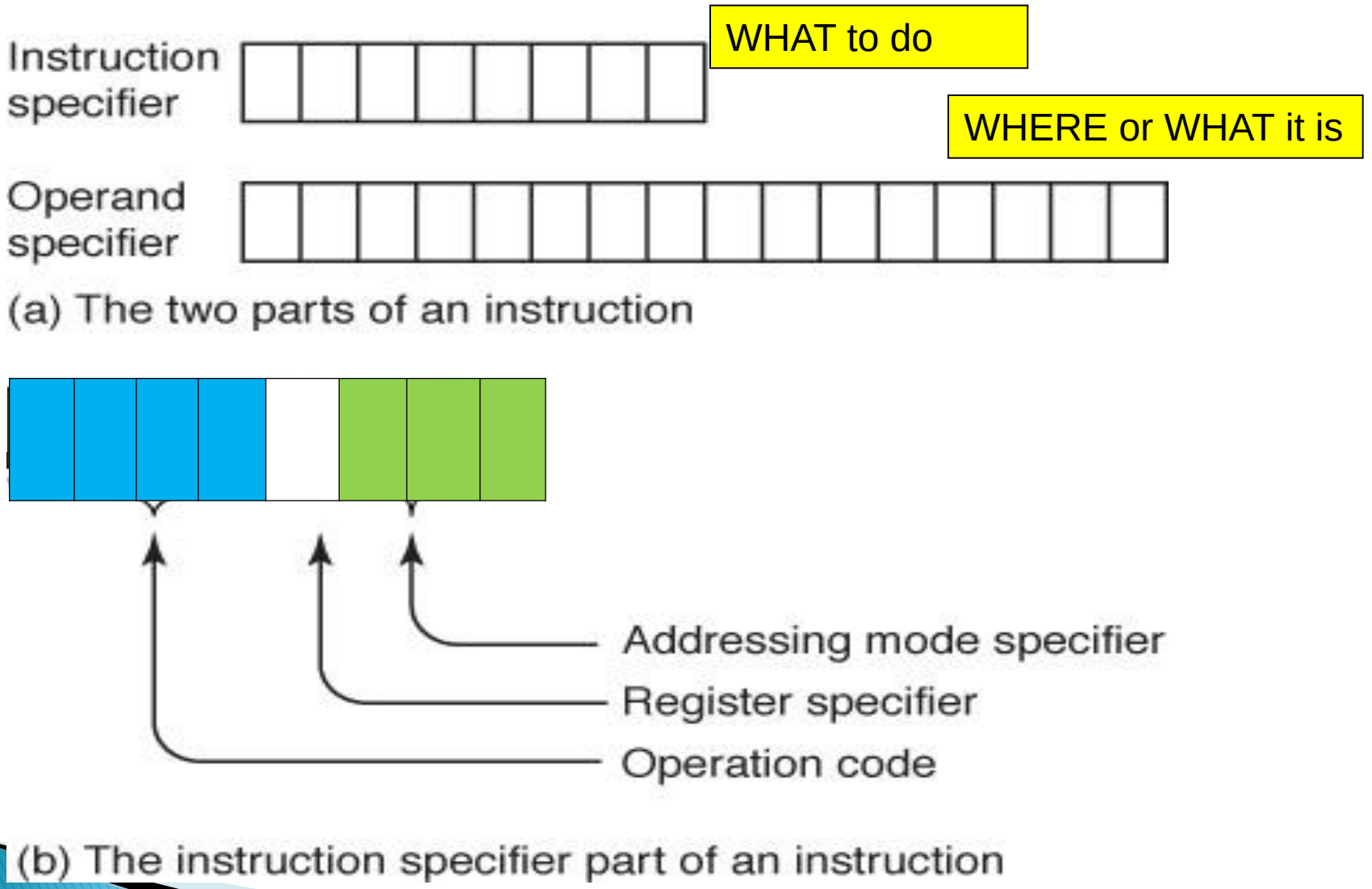
- Specify the OPERATION required e.g.
  - STOP the program
  - ADD values
  - STORE a value in memory
  - FIND something in memory
- Specify WHERE the action is to take place e.g.
  - Which register
- Specify WHERE the value is to be found or stored in memory…or specify the value itself.

Instruction Format



(a) The two parts of an instruction

Instruction Format



Instruction Format

Operation code

Specifies which instruction is to be carried out

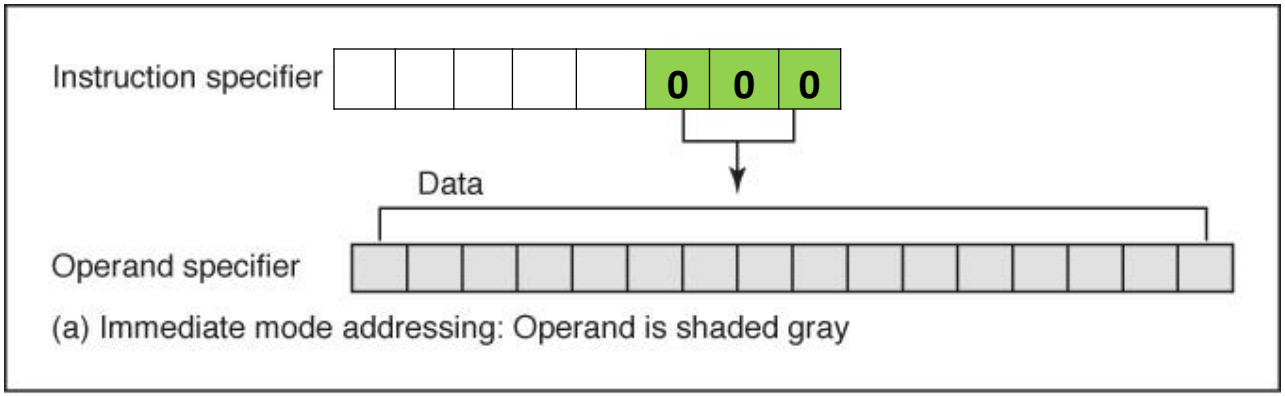
Register specifier

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

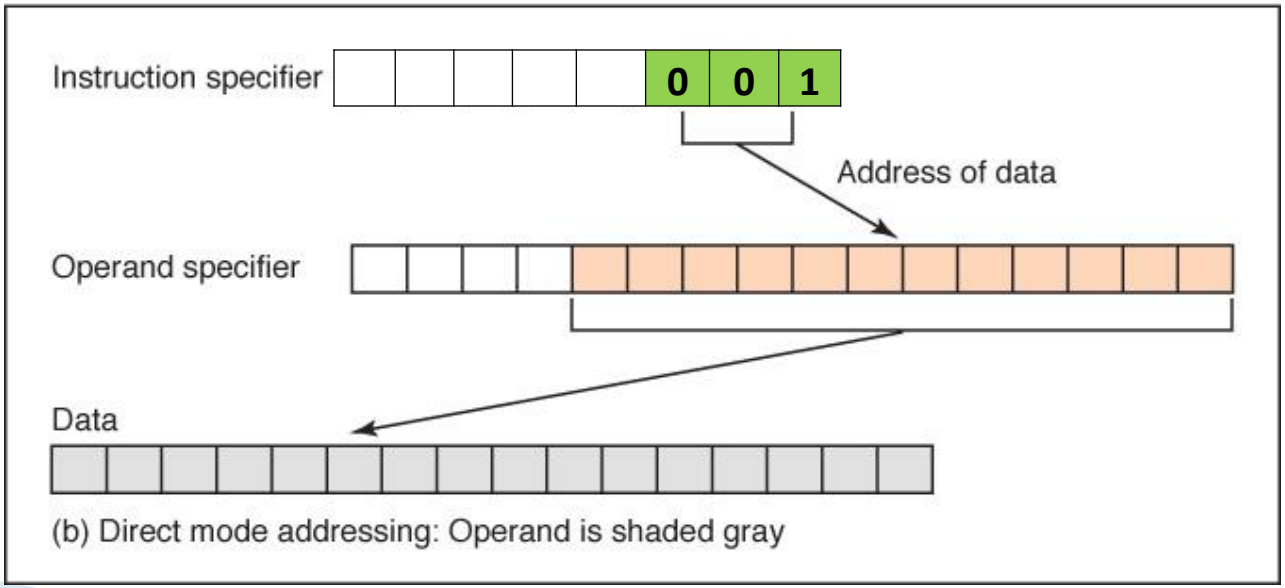
Addressing-mode specifier

Says how to interpret the operand part of the instruction

Instruction Format



(a) Immediate mode addressing: Operand is shaded gray



(b) Direct mode addressing: Operand is shaded gray

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

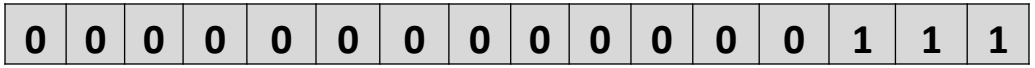
Some Sample Instructions

0000 STOP EXECUTION

1100 LOAD OPERAND into the A REGISTER

- Immediate addressing case:
  Instruction specifier:

  Operand specifier:



  What happens?

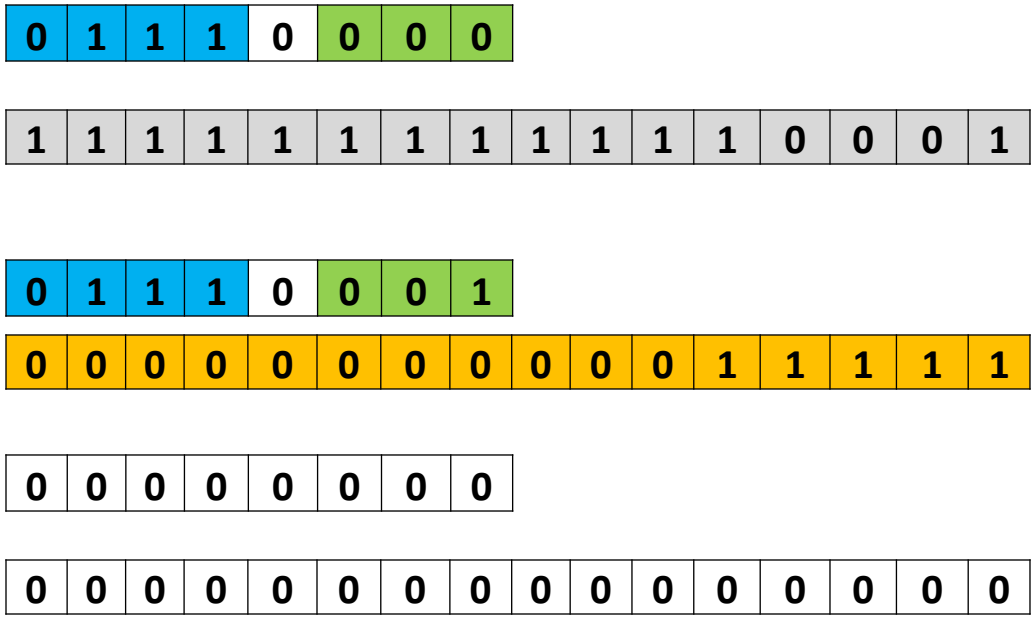
- Direct addressing case:
  Instruction specifier:

  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

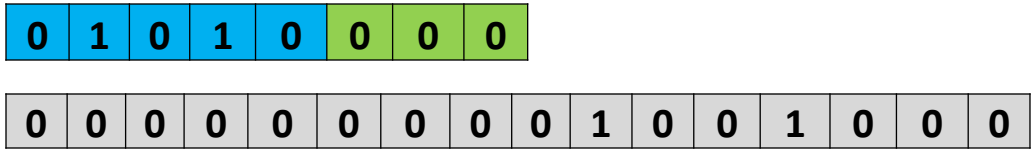
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1

0	1	0	0	1	0	0	1

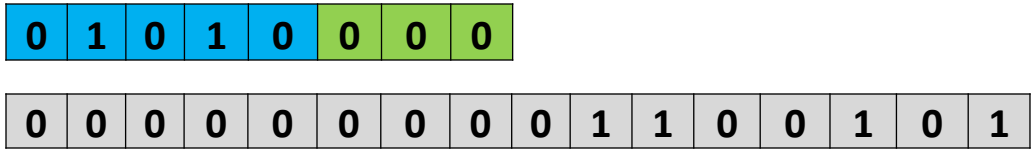
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1

A Program to ask for Help!

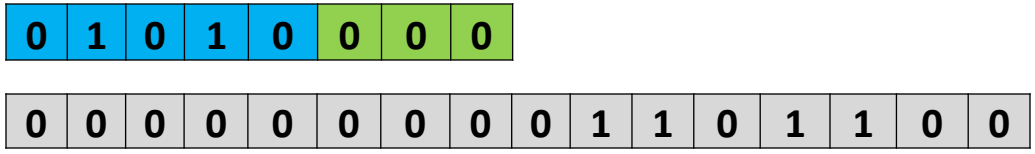
Opcode	Meaning of Instruction
0000	Stop execution
1100	Load the operand into the A register
1110	Store the contents of the A register into operand
0111	Add the operand to the A register
1000	Subtract the operand from the A register
01001	Character input to the operand
01010	Character output from the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



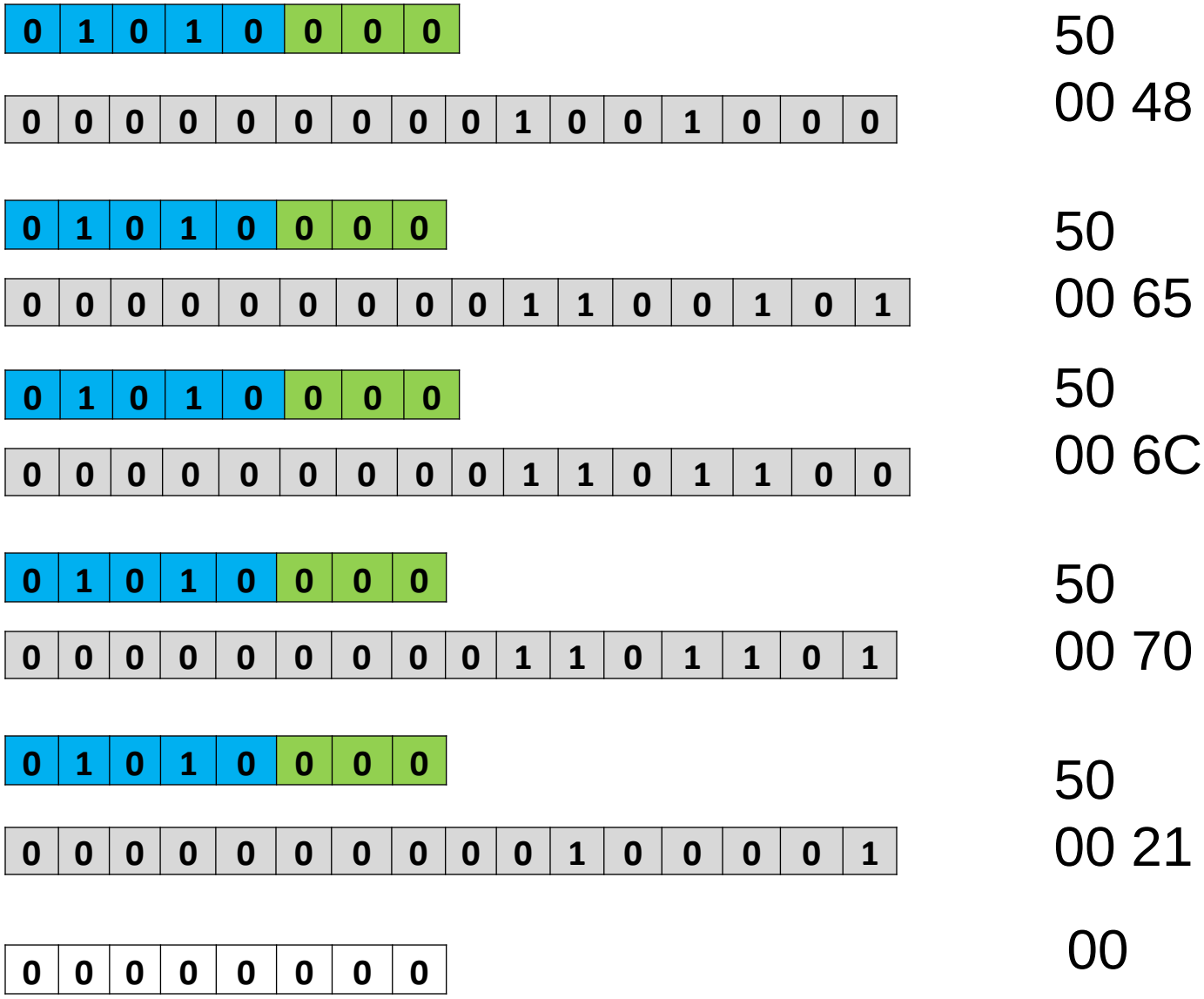
50
00 48



50
00 65



50
00 6C


0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0
50
00 48
0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1
50
00 65
0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 0
50
00 6C
0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 1 1 0 1 1 0 1
50
00 70
0 1 0 1 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1
50
00 21
0 0 0 0 0 0 0 0
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

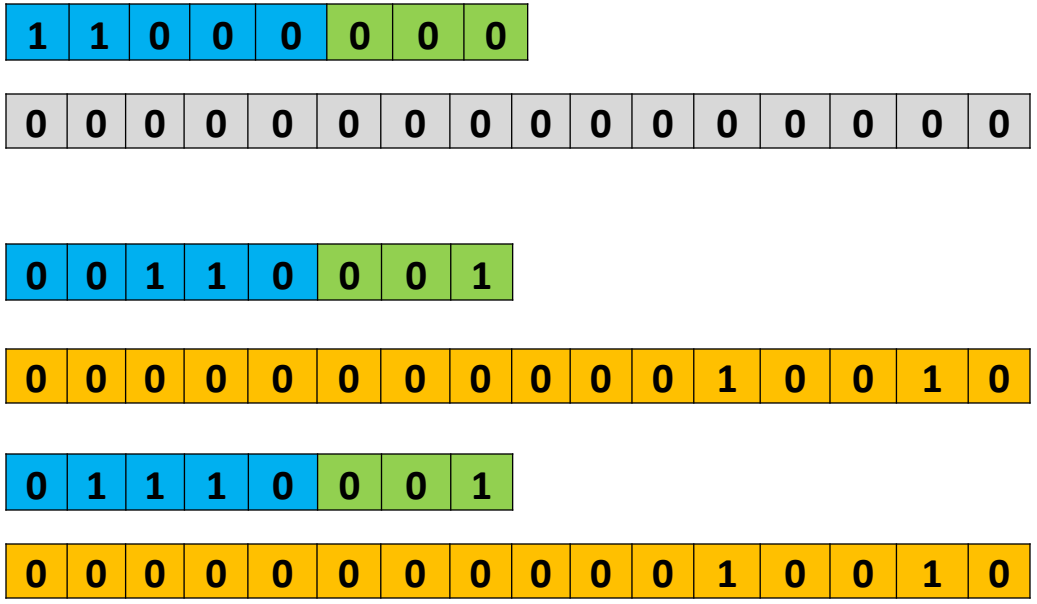
Run Object Code

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

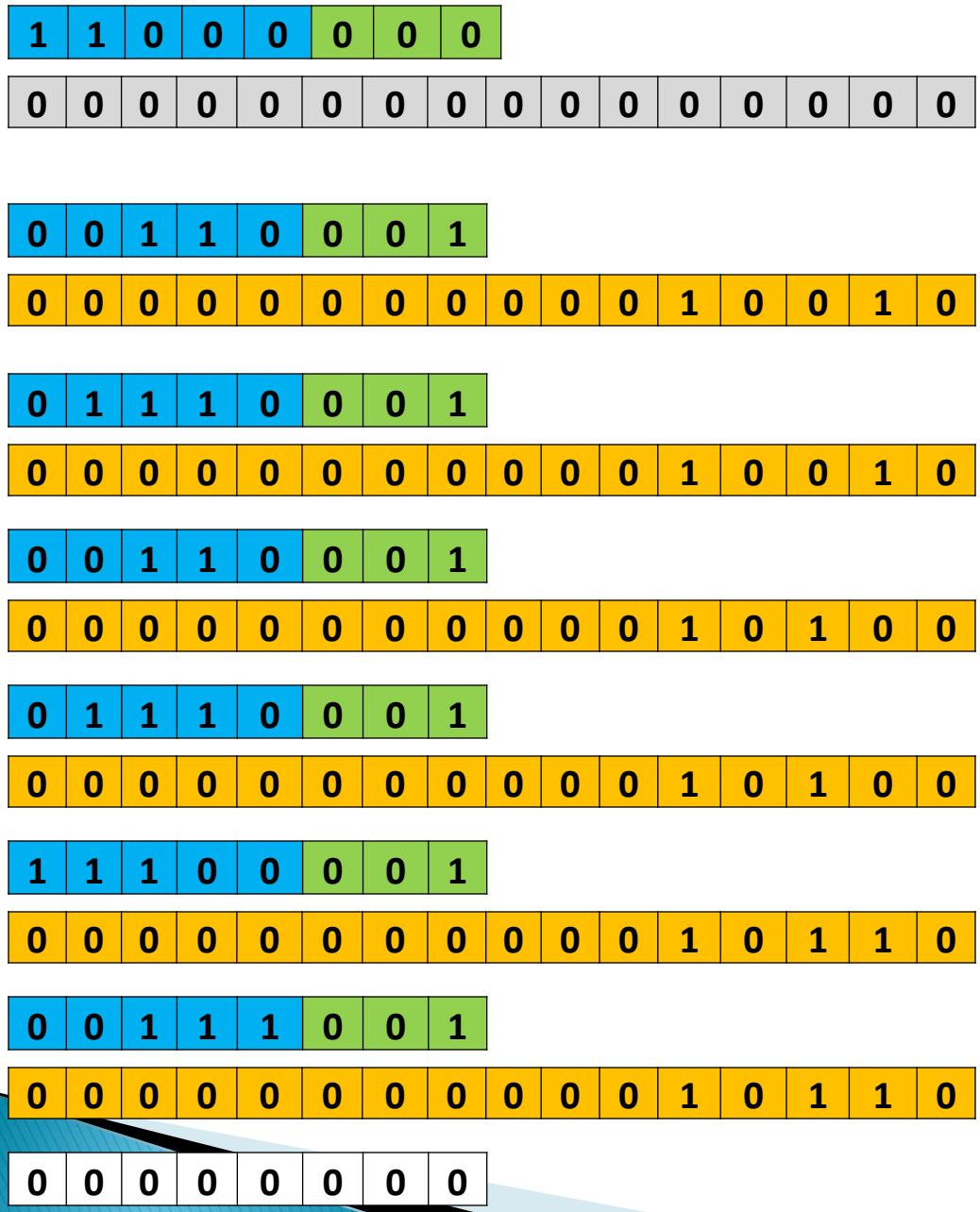
Opcode	Meaning of Instruction
0000	Stop execution
1100	Load the operand into the A register
1110	Store the contents of the A register into operand
0111	Add the operand to the A register
1000	Subtract the operand from the A register
00110	**Read in a decimal number**
00111	**Read out a decimal number**

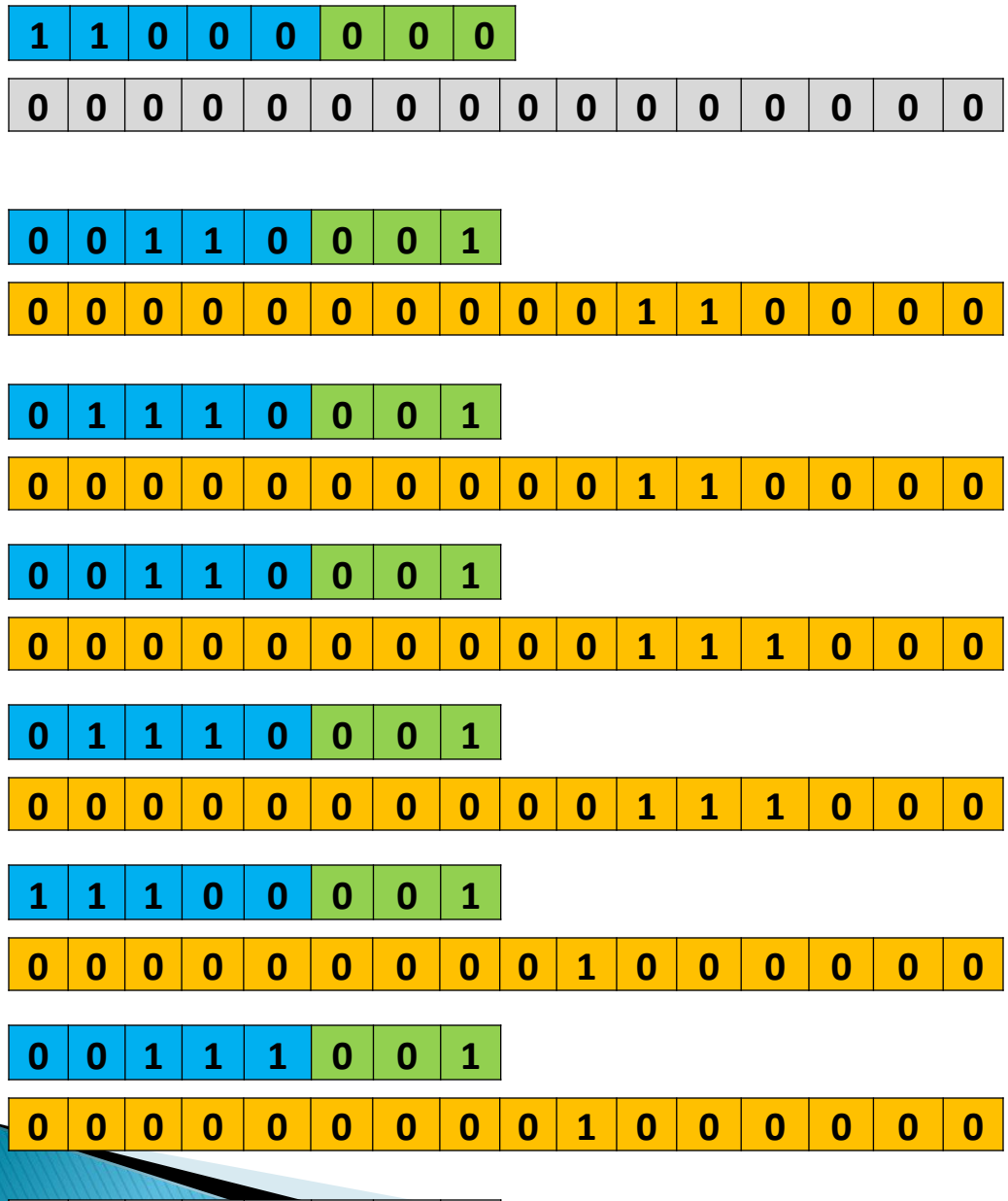


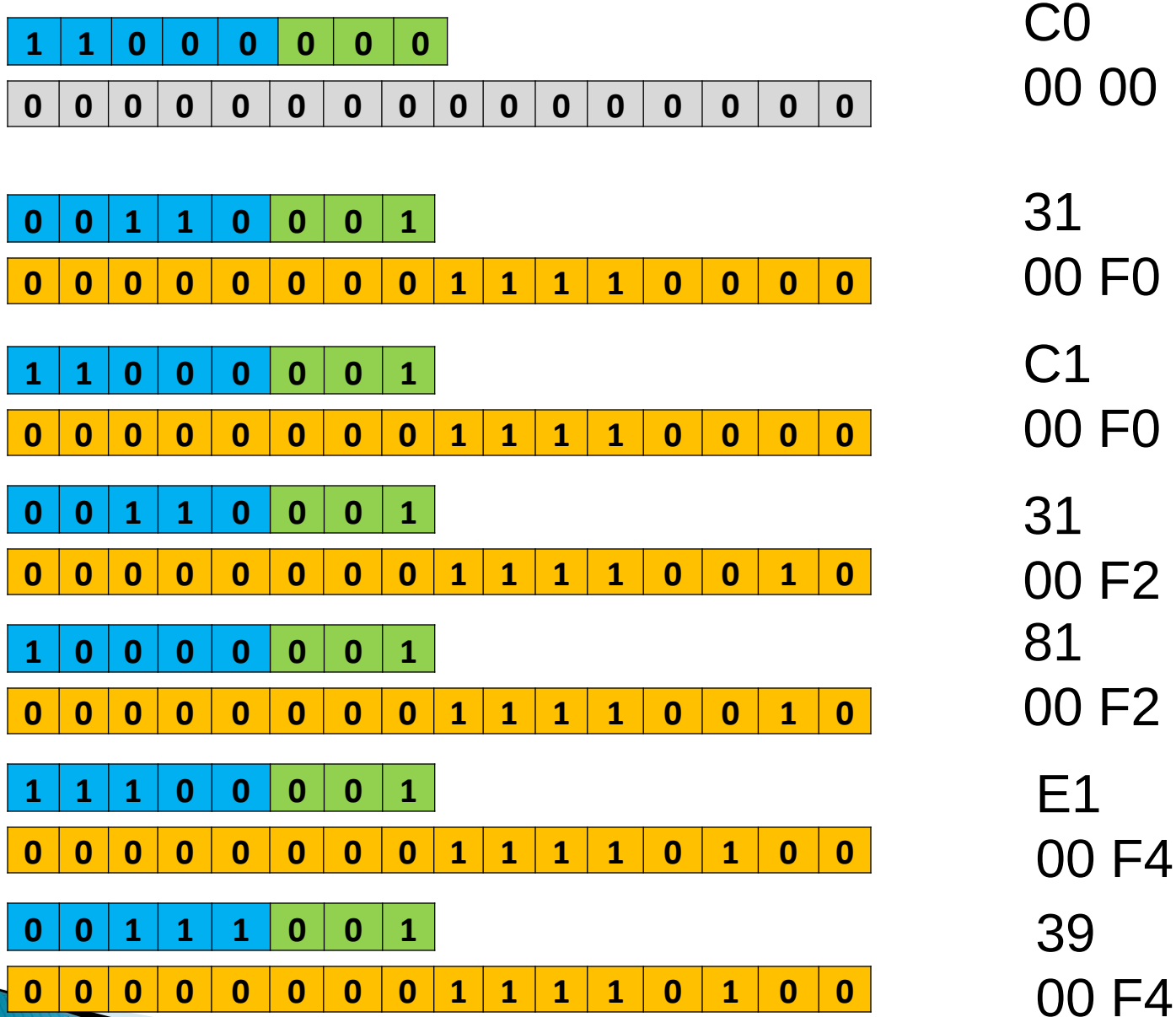
C0
00 00

31
00 12

71
00 12


1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
C0
00 00
0 0 1 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0
31
00 12
0 1 1 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0
71
00 12
0 0 1 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
31
00 14
0 1 1 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0
71
00 14
1 1 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0
E1
00 16
0 0 1 1 1 0 0 1
0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0
39
00 16
0 0 0 0 0 0 0 0


1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
C0
00 00
0 0 1 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
31
00 30
0 1 1 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0
71
00 30
0 0 1 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0
31
00 38
0 1 1 1 0 0 0 1
0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0
71
00 38
1 1 1 0 0 0 0 1
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
E1
00 40
0 0 1 1 1 0 0 1
0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
39
00 40
0 0 0 0 0 0 0 0


1 1 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
C0
00 00
0 0 1 1 0 0 0 1
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
31
00 F0
1 1 0 0 0 0 0 1
0 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0
C1
00 F0
0 0 1 1 0 0 0 1
0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0
31
00 F2
1 0 0 0 0 0 0 1
0 0 0 0 0 0 0 0 1 1 1 1 0 0 1 0
81
00 F2
1 1 1 0 0 0 0 1
0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0
E1
00 F4
0 0 1 1 1 0 0 1
0 0 0 0 0 0 0 0 1 1 1 1 0 1 0 0
39
00 F4
0 0 0 0 0 0 0 0

Assembly Language

Assembly language

A language that uses mnemonic codes to represent machine-language instructions

Assembler

A program that reads each of the instructions in mnemonic form and translates it into the machine-language equivalent

Assembly Process

Pep/8 Assembly Language

- Uses mnemonics for the codes to provide the machine codes we want
- Groups certain standard instruction sets into one mnemonic
- Has Assembler directives (or pseudo-operations) to instruct the assembler
- The next slide shows a mnemonic subset:

Mnemonic	Operand Mode specifier	Meaning of Instruction
Stop		Stop execution
LDA	0x008B,i	Load 008B into Register A
LDA	0x008B,d	Load the contents of location 008B into Register A
STA	0x008B,d	Store the contents of Register A into location 008B
ADDA	0x008B,i	Add 008B into Register A
ADDA	0x008B,d	Add the contents of location 008B to Register A
SUBA	0x008B,i	Subtract 008B from Register A
SUBA	0x008B,d	Subtract the contents of location 008B from Register A
BR		Branch to the location specified in the operand specifier
CHARI	0x008B,d	Read a character and store it in location 008B
CHARO	0x008B,i	Write the character 8B
CHARO	0x008B,d	Write the character stored in location 008B
DECI	0x008B,d	Read a decimal number and store it in location 008B
DECO	0x008B,i	Write the decimal number 139 (8B in hex)
DECO	0x008B,d	Write the decimal number stored in location 008B

Assembler Directives

Pseudo-op	Argument	Meaning of Instruction
.ASCII	"Str\x00"	Represents a string of ASCII bytes
.BLOCK	Number of bytes	Creates a block of bytes
.WORD	Value	Creates a word and stores a value in it
.END		Signals the end of the assembly language list

Program for Help!

```
CHARO 0x0048,i; Output an 'H'
CHARO 0x0065,i; Output an 'e'
CHARO 0x006C,i; Output an 'l'
CHARO 0x0070,i; Output a 'p'
CHARO 0x0021,i; Output an '!'
STOP
.END
```

Assembler Program to Add Three Numbers

```
                BR main ;
Branch around data
sum:        .WORD       0x0000 ;
Set up word ‘sum’ with zero value
num1:       .BLOCK 2 ;
Set up a two byte block for 'num1'
num2:       .BLOCK 2 ;
Set up a two byte block for 'num2'
num3:       .BLOCK 2 ;
Set up a two byte block for 'num3'
```

```
Main:       LDA sum,d ;
Start of Program. Load zero into Accumulator
            DECI num1,d ;
Read and store 'num1'
            ADDA num1,d ;
Add 'num1' to the Accumulator
            DECI num2,d ;
Read and store 'num2'
            ADDA num2,d ;
Add 'num2' to the Accumulator
            DECI num3,d ;
Read and store 'num3'
```

```
        ADDA num3,d ;
Add 'num3' to the Accumulator
        STA sum,d ;
Store the accumulator into 'sum'
        DECO sum,d ;
Output 'sum'
        STOP
        .END
```

```
Main:   LDA sum,d ;
        DECI num1,d ;
        ADDA num1,d ;
        DECI num2,d ;
        ADDA num2,d ;
        DECI num3,d ;
        ADDA num3,d ;
        STA sum,d ;
        DECO sum,d ;
        STOP
        .END
```

Assembler Program to Add Three Numbers neatly

```
            BR main ;
Branch around data
sum:        .WORD       0x0000 ;
Set up word ‘sum’ with zero value
num:        .BLOCK 2 ;
Set up a two byte block for 'num'
```

```
Main:   LDA sum,d ;
        DECI num,d ;
        ADDA num,d ;
        DECI num,d ;
        ADDA num,d ;
        DECI num,d ;
        ADDA num,d ;
        STA sum,d ;
        DECO sum,d ;
        STOP
        .END
```

Branching

- We have already seen that we can Branch to a different point in the program (actually directly set the Program Counter)
- We can also do conditional branches.
- This allows us to implement test statements and loops – the most valuable features of the computing process.

Branching Instructions

Mnemonic	Operand Mode specifier	Meaning of Instruction
BR	i	Branch to the location specified in the operand specifier
BRLT	i	Set PC to the operand if the A register is less than zero
BREQ	i	Set PC to the operand if the A register is equal to zero

Sum numbers in a loop

```
        br    main; Branch around the data
sum:    .word 0x0000; Reserve 0 WORD for SUM
num:    .block 2; Reserve BLOCK for num
limit:  .block 2; Reserve BLOCK for limit
```

```
main:   deci   limit,d; Read limit for numbers
loop:   deci   num,d;   Read number to add
        lda    sum,d;   Load sum so far into A
        adda   num,d;   Add the new number
        sta    sum,d;   Store answer in SUM
        lda    limit,d; Check if we've read all
        suba   1,i;     Subtract 1 from LIMIT
        sta    limit,d; Store value of LIMIT
        brgt   loop;    Continue loop if not end
quit:   deco   sum,d;   Write out answer
        stop
       .end
```

Sum numbers in a loop with test

```
        br    main; Branch around the data
sum:    .word 0x0000; Reserve 0 WORD for SUM
num:    .block 2; Reserve BLOCK for num
limit:  .block 2; Reserve BLOCK for limit
counter: .word 0x0000; Reserve 0 WORD for COUNTER

error: .ASCII “Error: Limit not positive”; Error message
```

```
main:   deci    limit,d;  Read limit for addition
        lda     limit,d;  Load limit into A
        brgt    loop;     Go on if positive
        stro    error,d;  Print error message
        stop;             Terminate program

loop:   as before
```