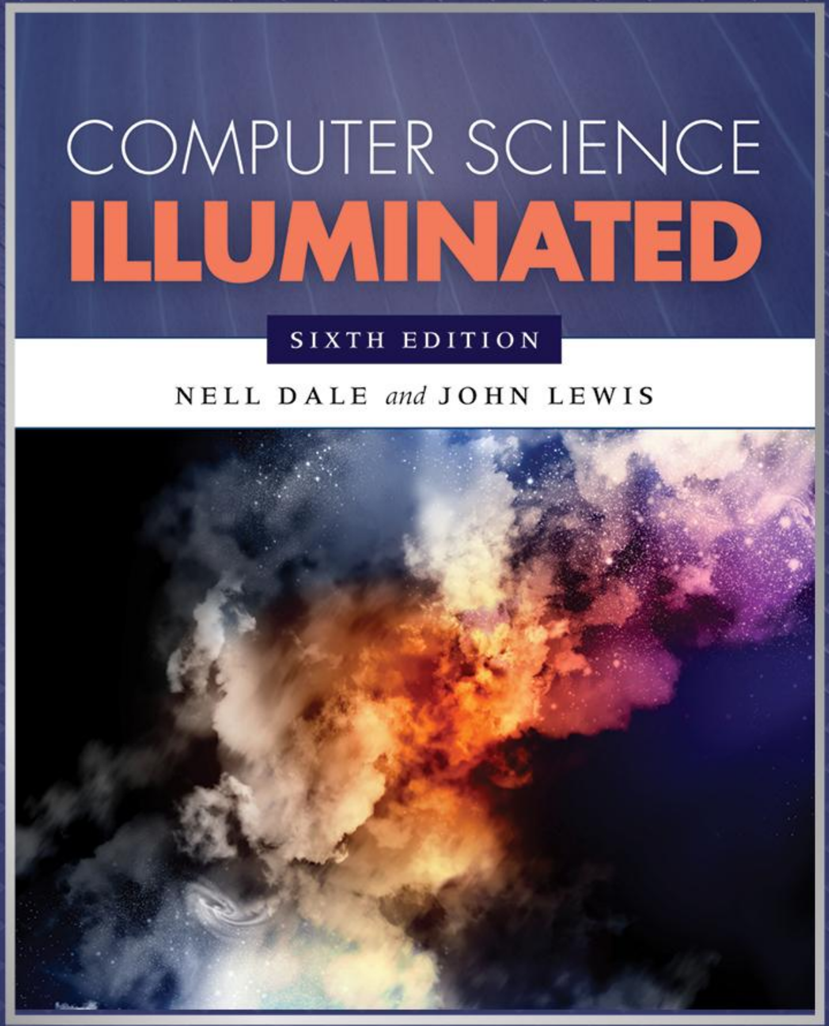
Chapter 6

Low-Level Programming Languages and Pseudocode



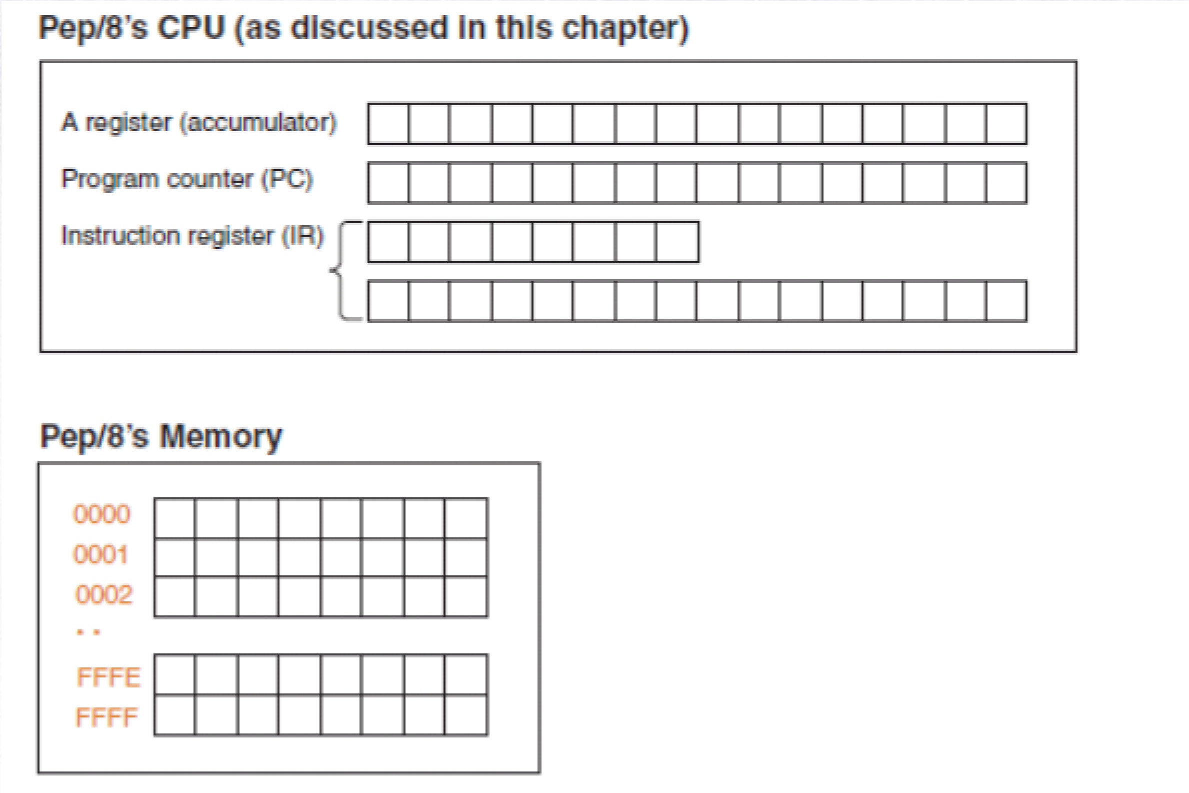
Assembly Programming

A quick recap on the Pep/8 machine:

- Program counter (PC) (contains the address of the next instruction to be executed).
- Instruction register (IR) (contains a copy of instruction being executed).
- Accumulator (A register for holding results and data).

The memory unit is made up of 65,636 bytes.

Assembly Programming



Assembly Programming

Machine code:

- Low level instructions given in binary
- Operates directly on hardware
- Very specific operations
- Not very friendly to write or read

Pep/8 Simulator

Pep8/Simulator

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

To run a machine code program:

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

Assembly Language

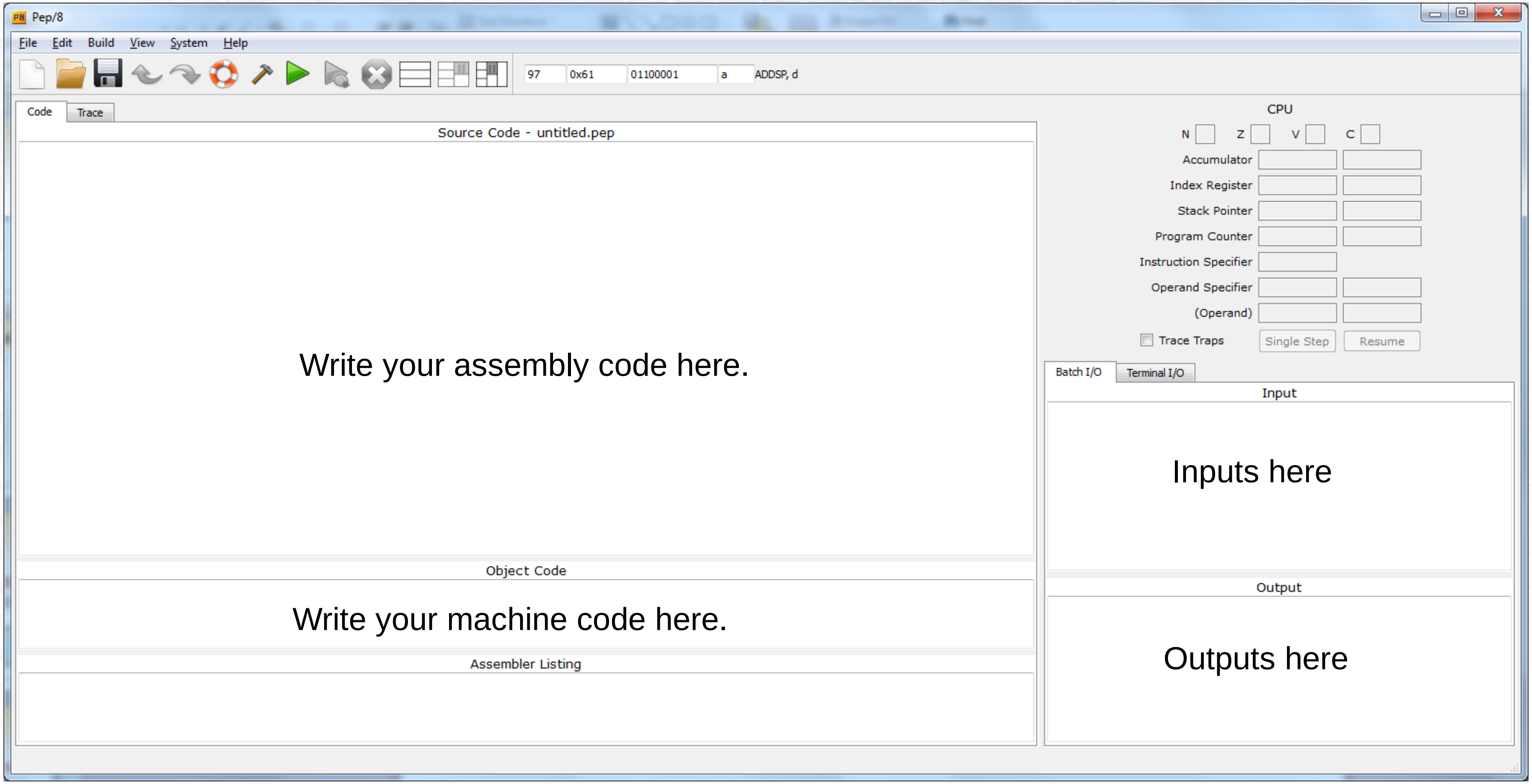
Assembly language

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

Pep/8 Simulator



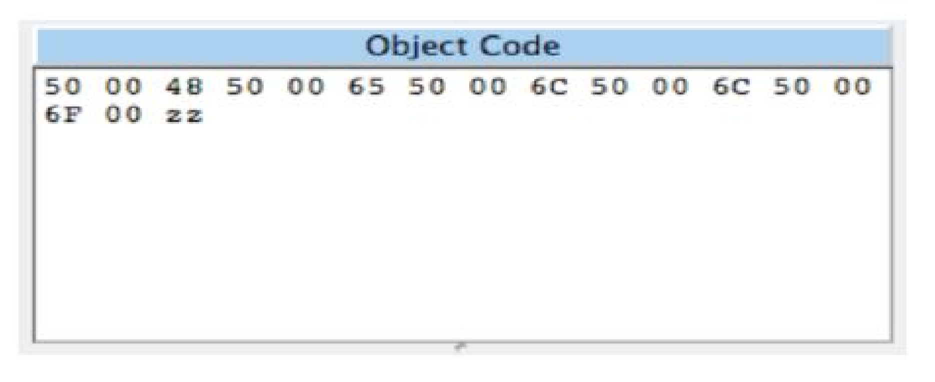
Write your assembly code here.

Write your machine code here.

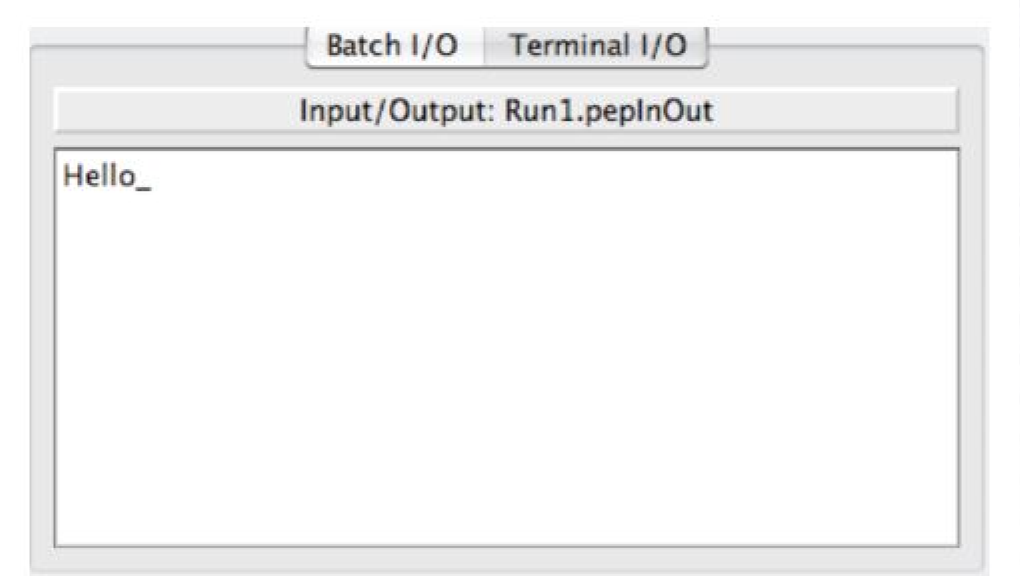
Inputs here

Outputs here

Pep/8 Simulator



```
50 00 48 50 00 65 50 00 6C 50 00 6C 50 00
6F 00 zz
```

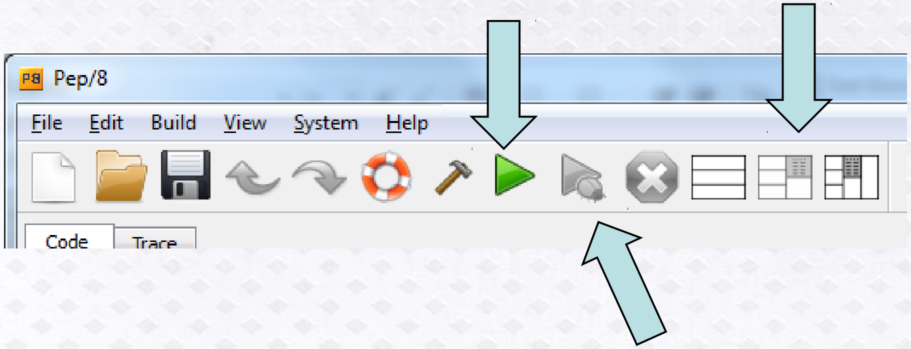


```
Hello_
```

What are the "zz"s for?

Pep/8 Simulator

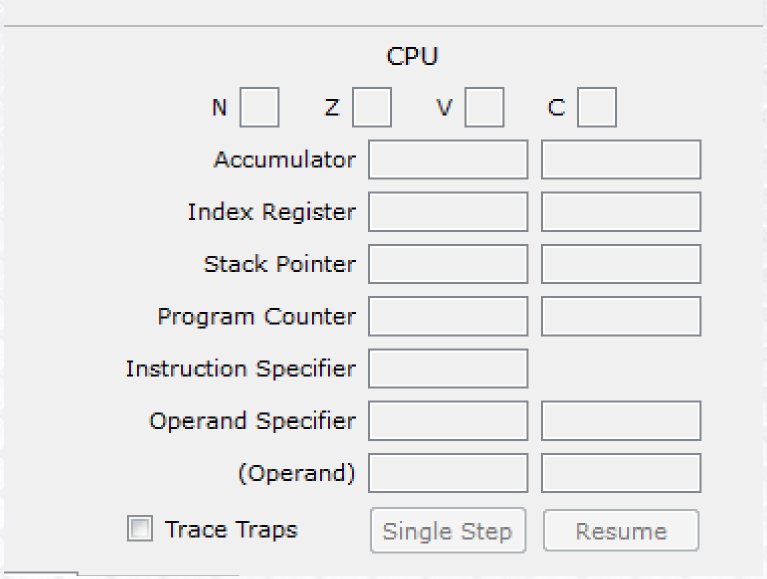
Execute *Change Views*



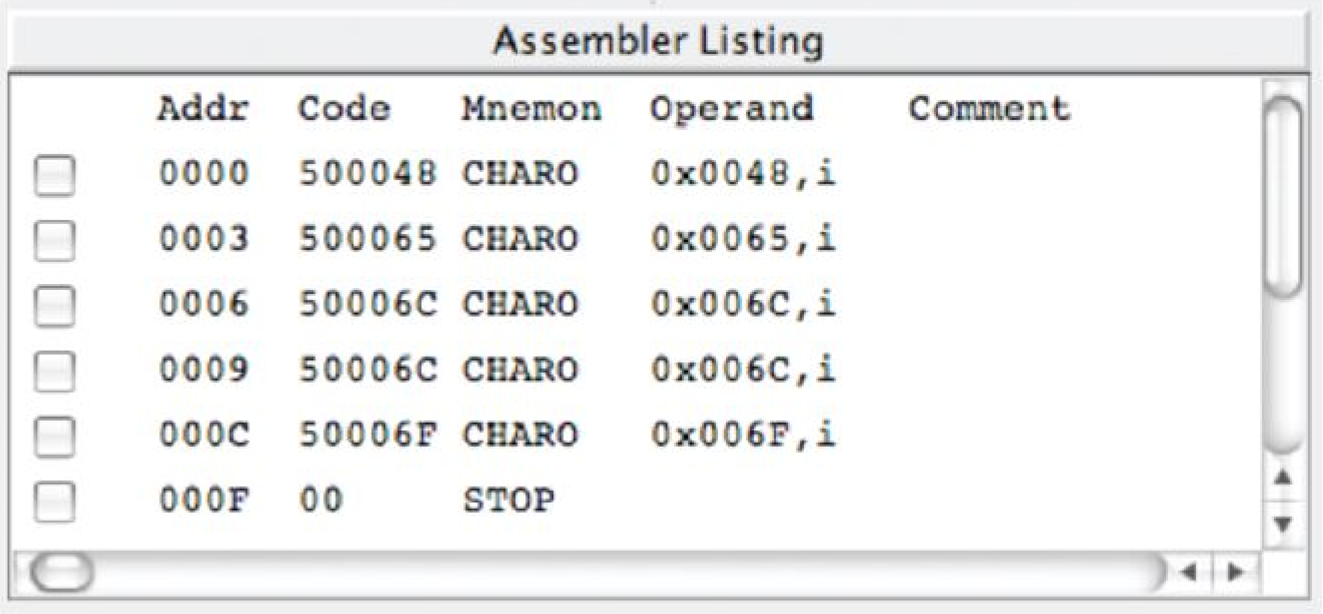
Build *Debug*

Your version may look just a bit *different.*

Pep/8 Simulator

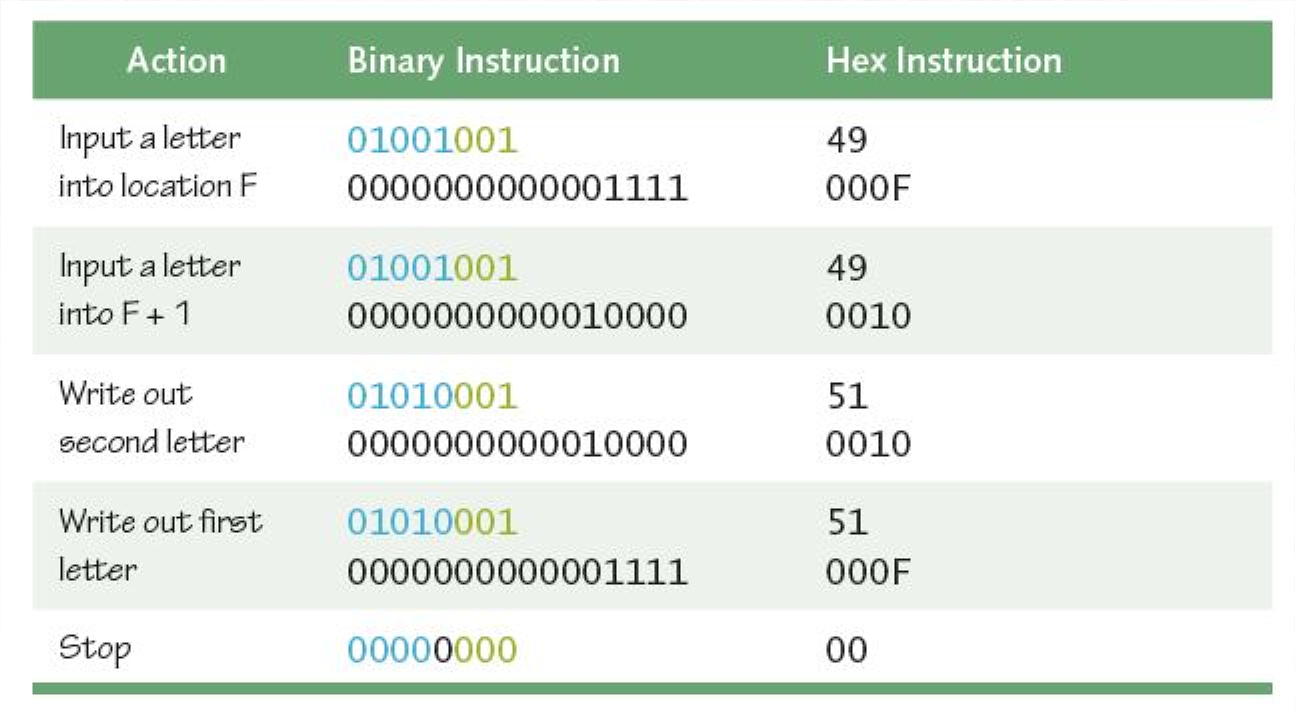


Pep/8 Assembly Language



```
     Addr  Code   Mnemon  Operand      Comment
□    0000  500048 CHARO   0x0048,i
□    0003  500065 CHARO   0x0065,i
□    0006  50006C CHARO   0x006C,i
□    0009  50006C CHARO   0x006C,i
□    000C  50006F CHARO   0x006F,i
□    000F  00     STOP
```

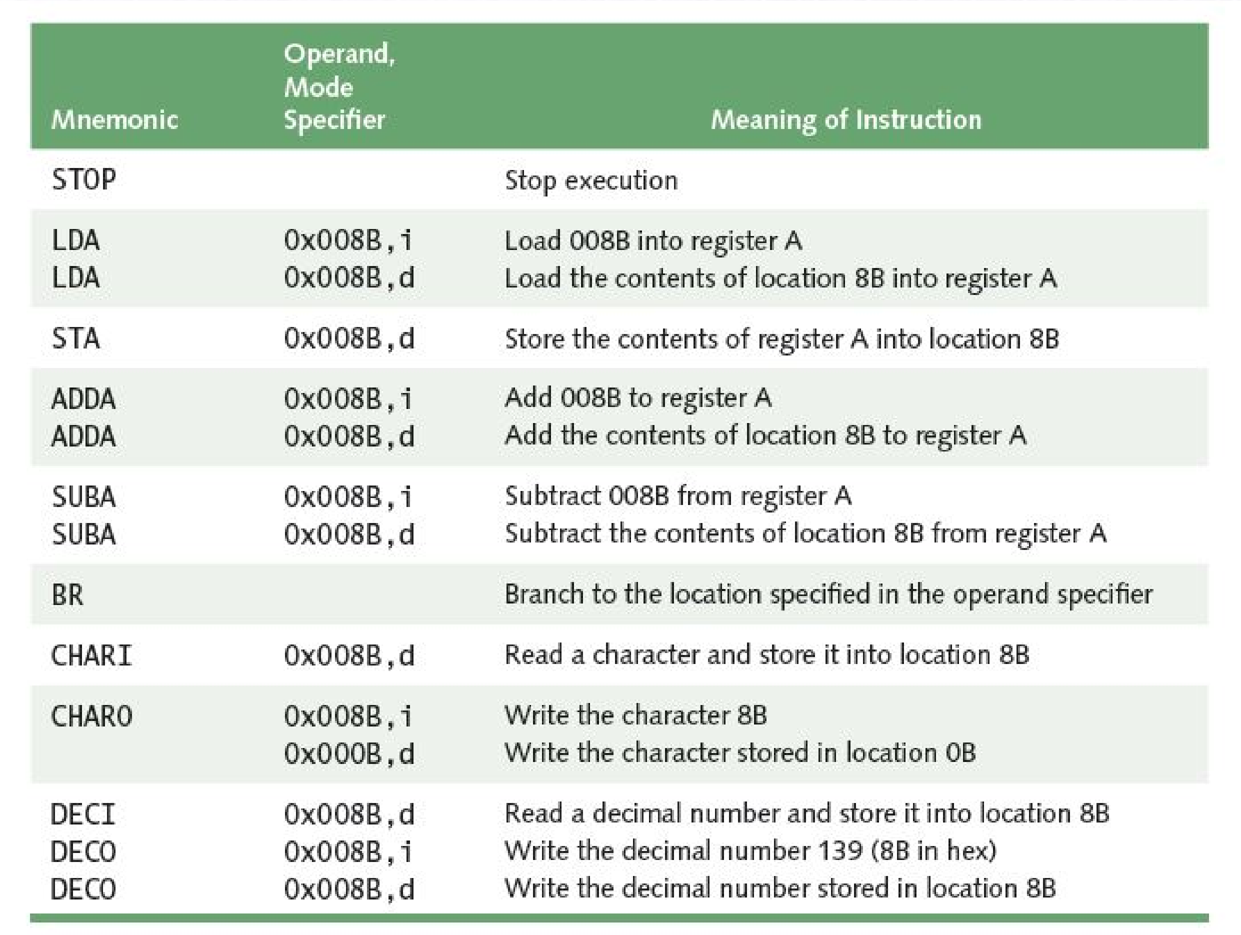
Pep/8 Simulator



Action	Binary Instruction	Hex Instruction
Input a letter into location F	01001001 0000000000001111	49 000F
Input a letter into F + 1	01001001 0000000000010000	49 0010
Write out second letter	01010001 0000000000010000	51 0010
Write out first letter	01010001 0000000000001111	51 000F
Stop	00000000	00

What does this program do?

Pep/8 Assembly Language

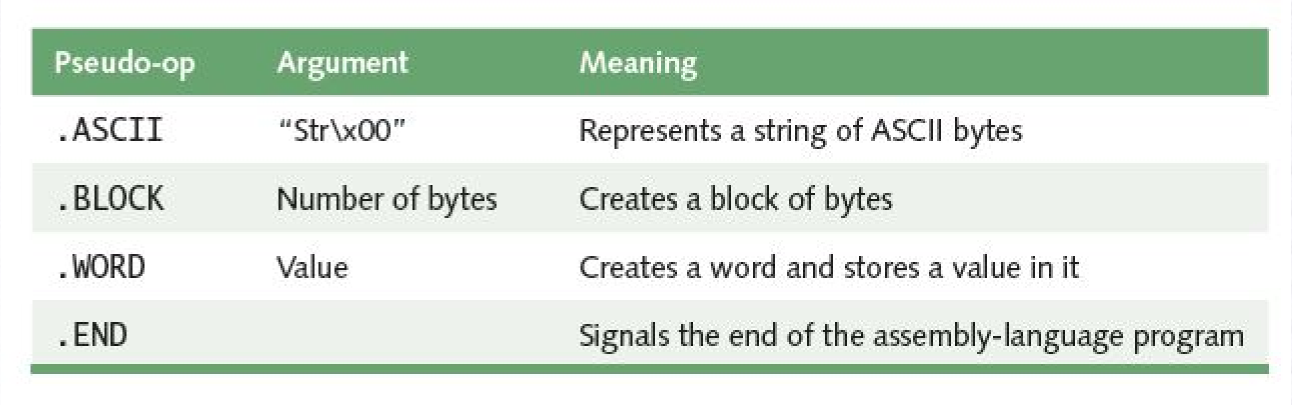


Mnemonic	Operand, Mode Specifier	Meaning of Instruction
STOP		Stop execution
LDA LDA	0x008B,i 0x008B,d	Load 008B into register A Load the contents of location 8B into register A
STA	0x008B,d	Store the contents of register A into location 8B
ADDA ADDA	0x008B,i 0x008B,d	Add 008B to register A Add the contents of location 8B to register A
SUBA SUBA	0x008B,i 0x008B,d	Subtract 008B from register A Subtract the contents of location 8B from register A
BR		Branch to the location specified in the operand specifier
CHARI	0x008B,d	Read a character and store it into location 8B
CHARO	0x008B,i 0x000B,d	Write the character 8B Write the character stored in location 0B
DECI DECO DECO	0x008B,d 0x008B,i 0x008B,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 location 8B

Remember the difference between immediate and direct addressing?

i : immediate
d: direct

Pep/8 Assembly Language



Pseudo-op	Argument	Meaning
.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 program

What is the difference between operations and pseudo operations?

Assembly Process

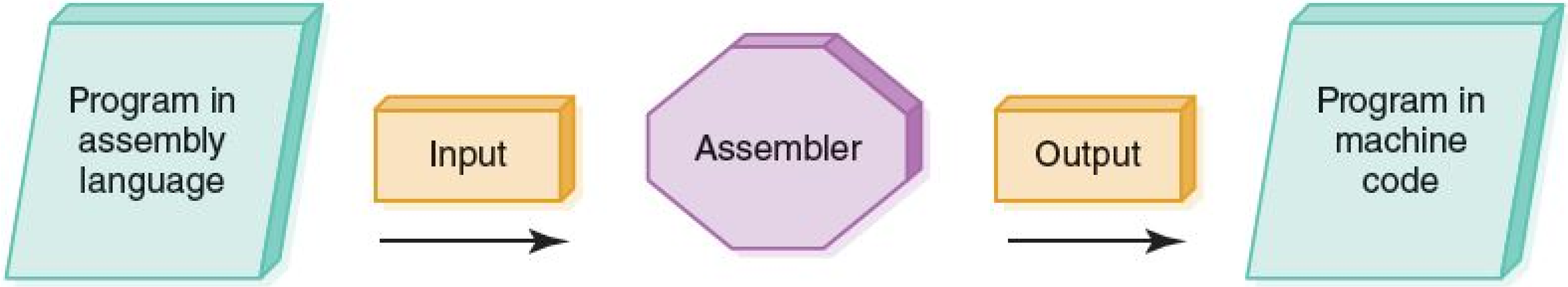


FIGURE 6.5 Assembly process

A New Program

Let’s try to implement the following:

- Load accumulator with 5.
- Store the value of accumulator in hex 60.
- Load accumulator with 6.
- Add contents of hex 60 to accumulator.
- Store result in hex 90.
- Stop.

How would you do it by hand?

Our Completed Program

In machine code:

```
11000000 00000000 00000101
11100001 00000000 01100000
11000000 00000000 00000110
01110001 00000000 01100000
11100001 00000000 10010000
00
```

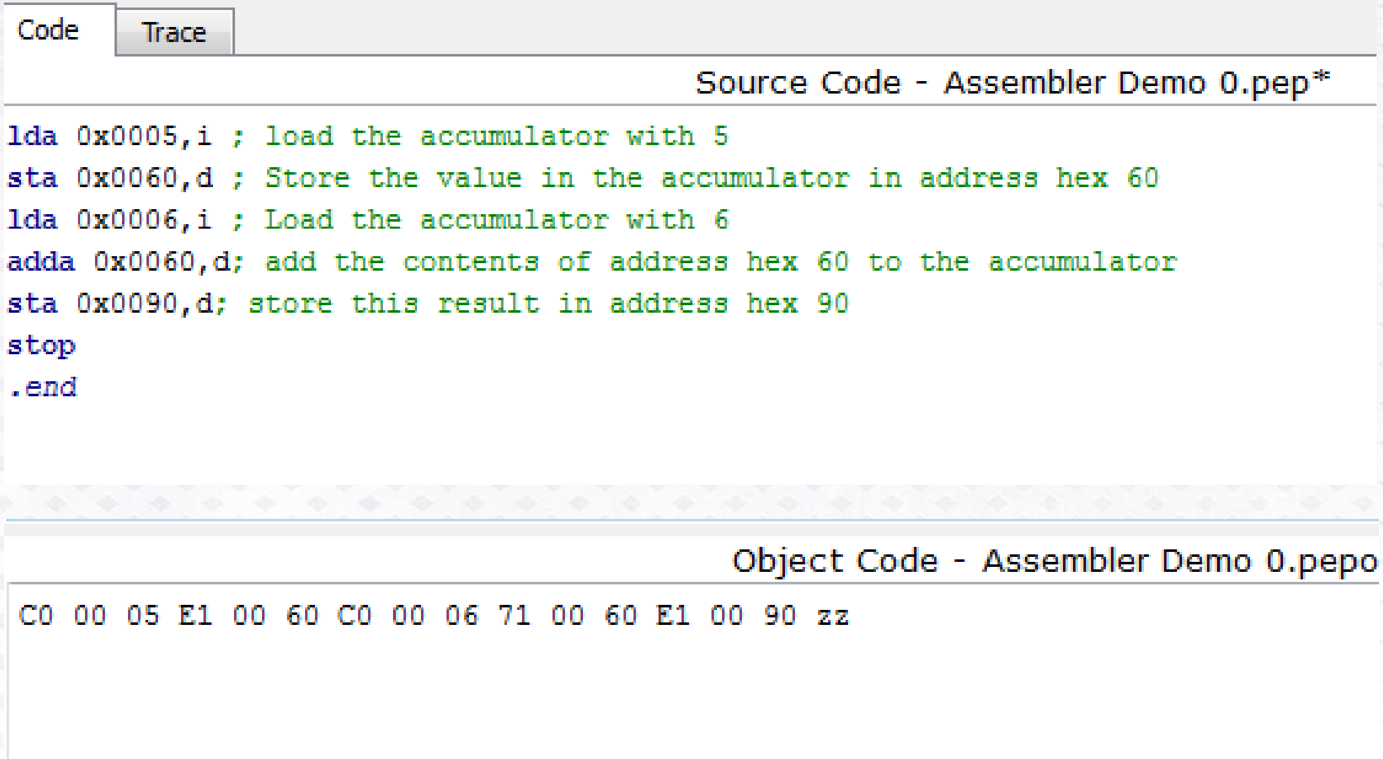
Our Completed Program

In machine code (as hexadecimal):

```
C0 00 05
E1 00 60
C0 00 06
71 00 60
E1 00 90
00
```

Our Completed Program

In Pep/8 assembly code:



Source Code - Assembler Demo 0.pep*

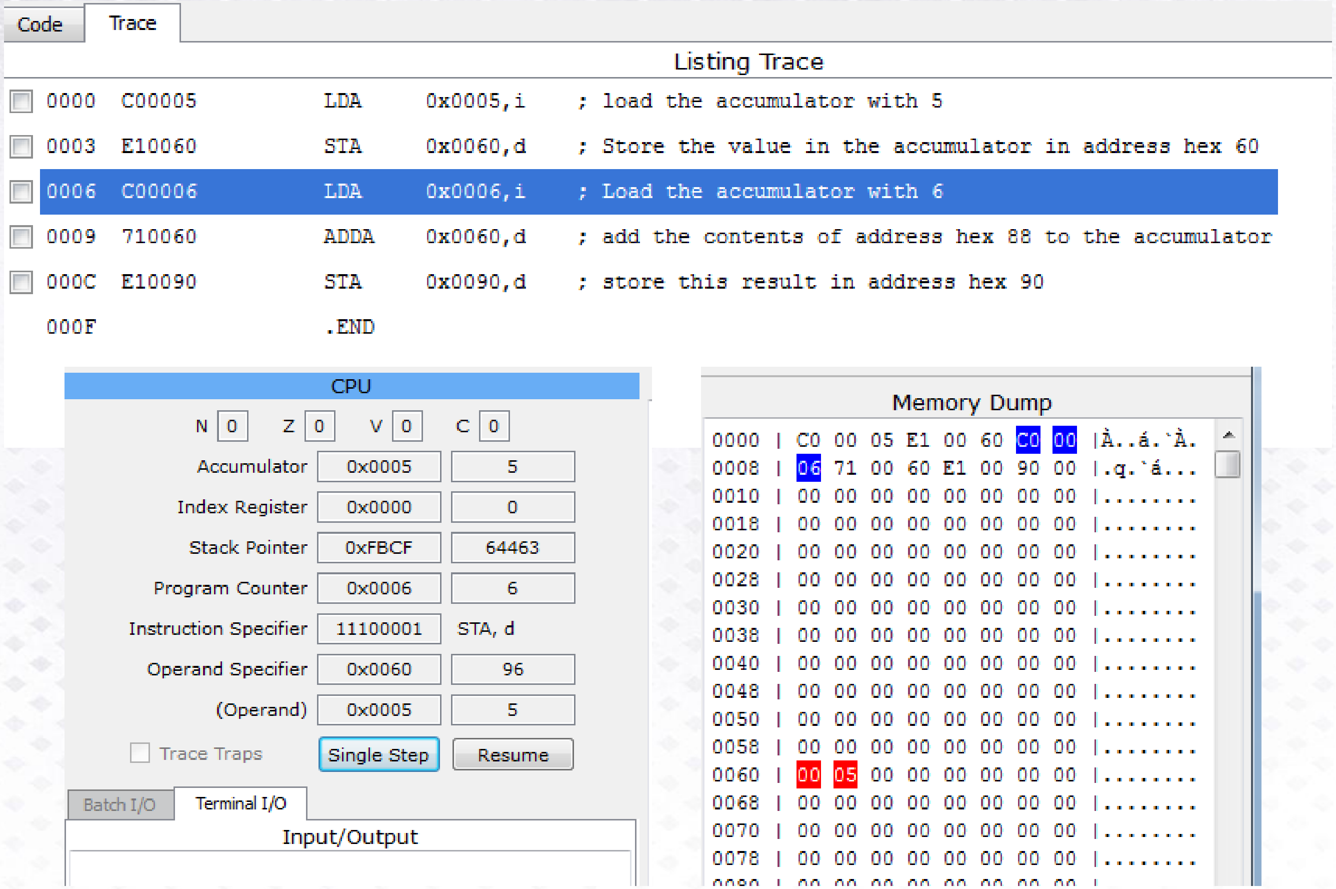
```
lda 0x0005,i ; load the accumulator with 5
sta 0x0060,d ; Store the value in the accumulator in address hex 60
lda 0x0006,i ; Load the accumulator with 6
adda 0x0060,d; add the contents of address hex 60 to the accumulator
sta 0x0090,d; store this result in address hex 90
stop
.end
```

Object Code - Assembler Demo 0.pepo

```
C0 00 05 E1 00 60 C0 00 06 71 00 60 E1 00 90 zz
```

A helpful hint: Debugging

Pep/8 also allows execution of programs one line at a time:



Listing Trace

```
□ 0000  C00005      LDA    0x0005,i    ; load the accumulator with 5
□ 0003  E10060      STA    0x0060,d    ; Store the value in the accumulator in address hex 60
□ 0006  C00006      LDA    0x0006,i    ; Load the accumulator with 6
□ 0009  710060      ADDA   0x0060,d    ; add the contents of address hex 88 to the accumulator
□ 000C  E10090      STA    0x0090,d    ; store this result in address hex 90
  000F              .END
```

CPU

N 0 Z 0 V 0 C 0

Accumulator	0x0005	5
Index Register	0x0000	0
Stack Pointer	0xFBCF	64463
Program Counter	0x0006	6
Instruction Specifier	11100001	STA, d
Operand Specifier	0x0060	96
(Operand)	0x0005	5

□ Trace Traps

Memory Dump

```
0000 | C0 00 05 E1 00 60 C0 00 |À..á.`À.
0008 | 06 71 00 60 E1 00 90 00 |.q.`á...
0010 | 00 00 00 00 00 00 00 00 |........
0018 | 00 00 00 00 00 00 00 00 |........
0020 | 00 00 00 00 00 00 00 00 |........
0028 | 00 00 00 00 00 00 00 00 |........
0030 | 00 00 00 00 00 00 00 00 |........
0038 | 00 00 00 00 00 00 00 00 |........
0040 | 00 00 00 00 00 00 00 00 |........
0048 | 00 00 00 00 00 00 00 00 |........
0050 | 00 00 00 00 00 00 00 00 |........
0058 | 00 00 00 00 00 00 00 00 |........
0060 | 00 05 00 00 00 00 00 00 |........
0068 | 00 00 00 00 00 00 00 00 |........
0070 | 00 00 00 00 00 00 00 00 |........
0078 | 00 00 00 00 00 00 00 00 |........
0080 | 00 00 00 00 00 00 00 00 |
```

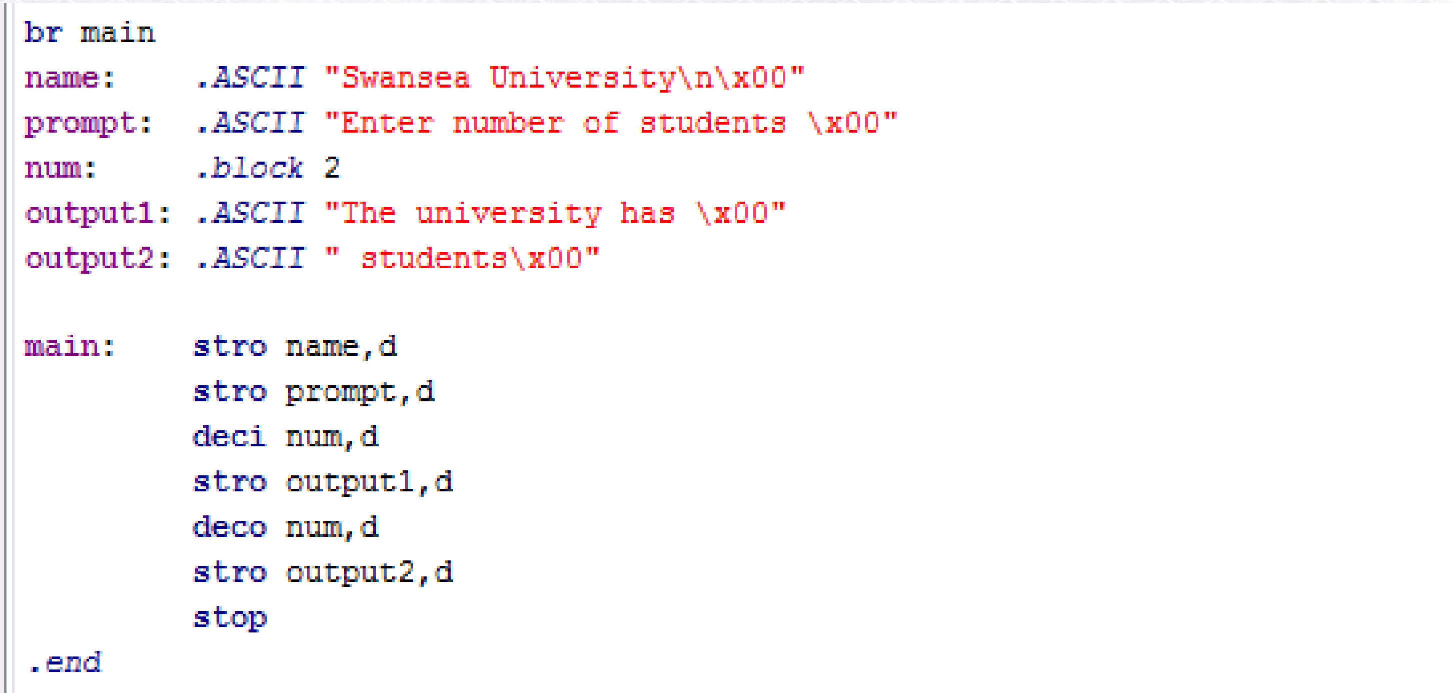
Example Assembly Programs: Printing Characters

The following program will print “Hello”:

```
CHARO 0x0048, i;   Output an 'H'
CHARO 0x0065, i;   Output an 'e'
CHARO 0x006C, i;   Output an 'l'
CHARO 0x006C, i;   Output an 'l'
CHARO 0x006F, i;   Output an 'o'
STOP
.END
```

Example Assembly Programs: Handling Strings

Strings are set up in memory using the .ASCII pseudo-op code:



```
br main
name:     .ASCII "Swansea University\n\x00"
prompt:   .ASCII "Enter number of students \x00"
num:      .block 2
output1:  .ASCII "The university has \x00"
output2:  .ASCII " students\x00"

main:     stro name,d
          stro prompt,d
          deci num,d
          stro output1,d
          deco num,d
          stro output2,d
          stop
.end
```

*Notice “stro” for string output – This highlights the Pep/8 naming conventions.

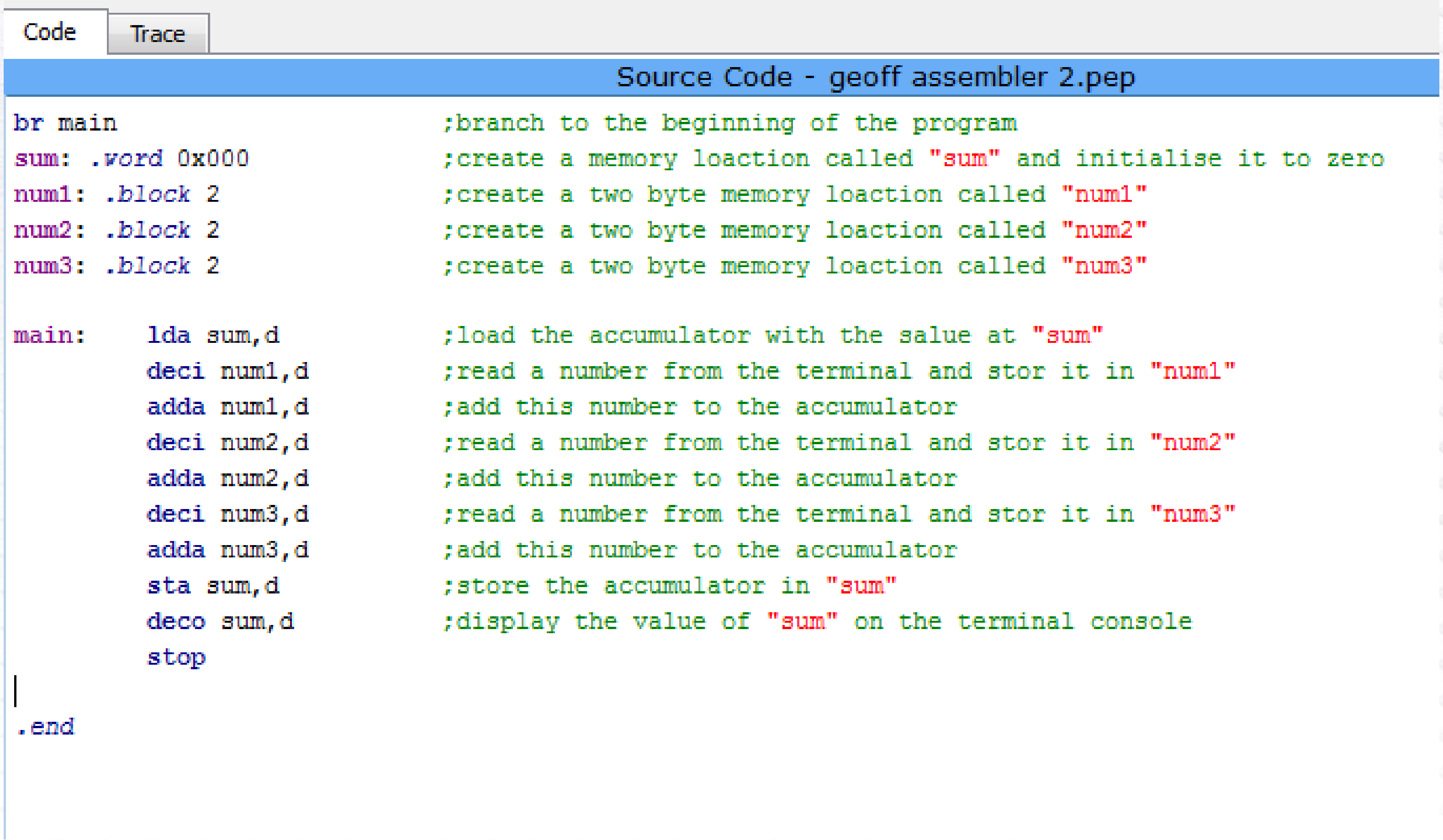
Example Assembly Programs: Variables

Rather than referring to memory by hex addresses, we can give it a name:

- .BLOCK – assigns one or more bytes of memory.
- .WORD – assigns memory, and initialises it to a set value.

(much the same as when we declare a variable in java)

Example Assembly Programs: Variables



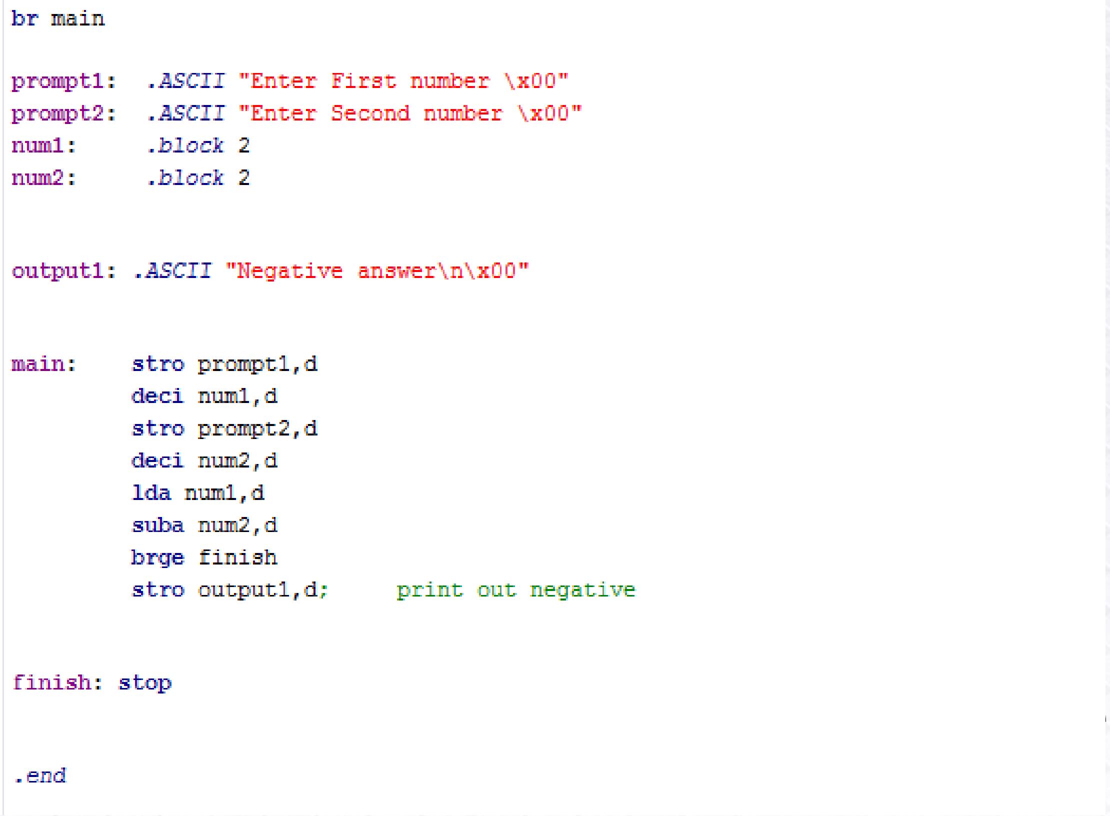
```
br main                       ;branch to the beginning of the program
sum: .word 0x000              ;create a memory loaction called "sum" and initialise it to zero
num1: .block 2                ;create a two byte memory loaction called "num1"
num2: .block 2                ;create a two byte memory loaction called "num2"
num3: .block 2                ;create a two byte memory loaction called "num3"

main:     lda sum,d           ;load the accumulator with the salue at "sum"
          deci num1,d         ;read a number from the terminal and stor it in "num1"
          adda num1,d         ;add this number to the accumulator
          deci num2,d         ;read a number from the terminal and stor it in "num2"
          adda num2,d         ;add this number to the accumulator
          deci num3,d         ;read a number from the terminal and stor it in "num3"
          adda num3,d         ;add this number to the accumulator
          sta sum,d           ;store the accumulator in "sum"
          deco sum,d          ;display the value of "sum" on the terminal console
          stop

.end
```

Example Assembly Programs: Branching

Branching constructs include:

- BRLT var; Branch to var if accumulator < 0.
- BRLE var; Branch to var if accumulator <= 0.
- BRGT var; Branch to var if accumulator > 0.
- BRGE var; Branch to var if accumulator >= 0.
- BREQ var; Branch to var if accumulator is 0.
- BRNE var; Branch to var if accumulator is not 0.

Example Assembly Programs: Branching



```
br main

prompt1:  .ASCII "Enter First number \x00"
prompt2:  .ASCII "Enter Second number \x00"
num1:     .block 2
num2:     .block 2


output1: .ASCII "Negative answer\n\x00"


main:    stro prompt1,d
         deci num1,d
         stro prompt2,d
         deci num2,d
         lda num1,d
         suba num2,d
         brge finish
         stro output1,d;     print out negative


finish: stop


.end
```

Perforn s
negativ

Ethical Issues

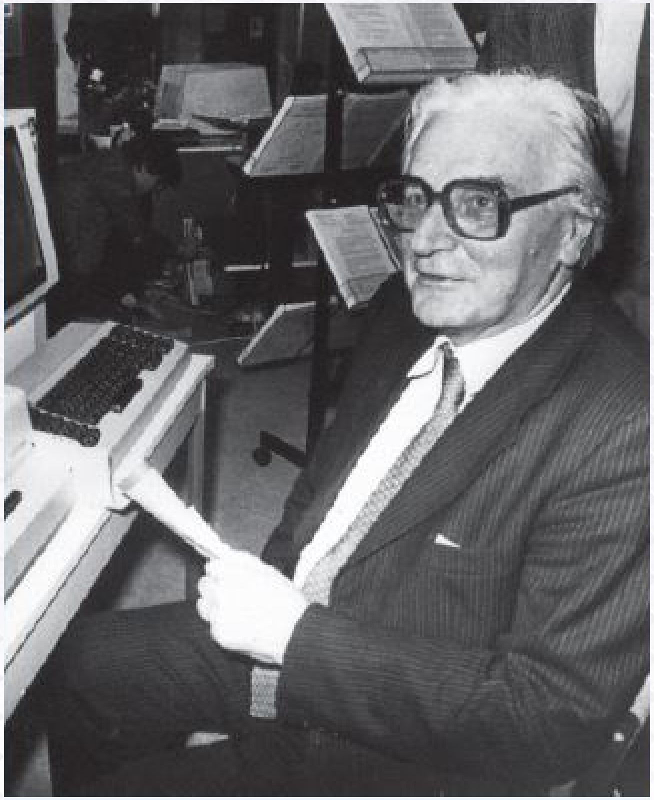
Software Piracy and Copyrighting

Have you ever "borrowed" software from a friend?

Have you ever "lent" software to a friend?

According to IDC, lowering software piracy by 10% over the next four years would create 500,000 jobs

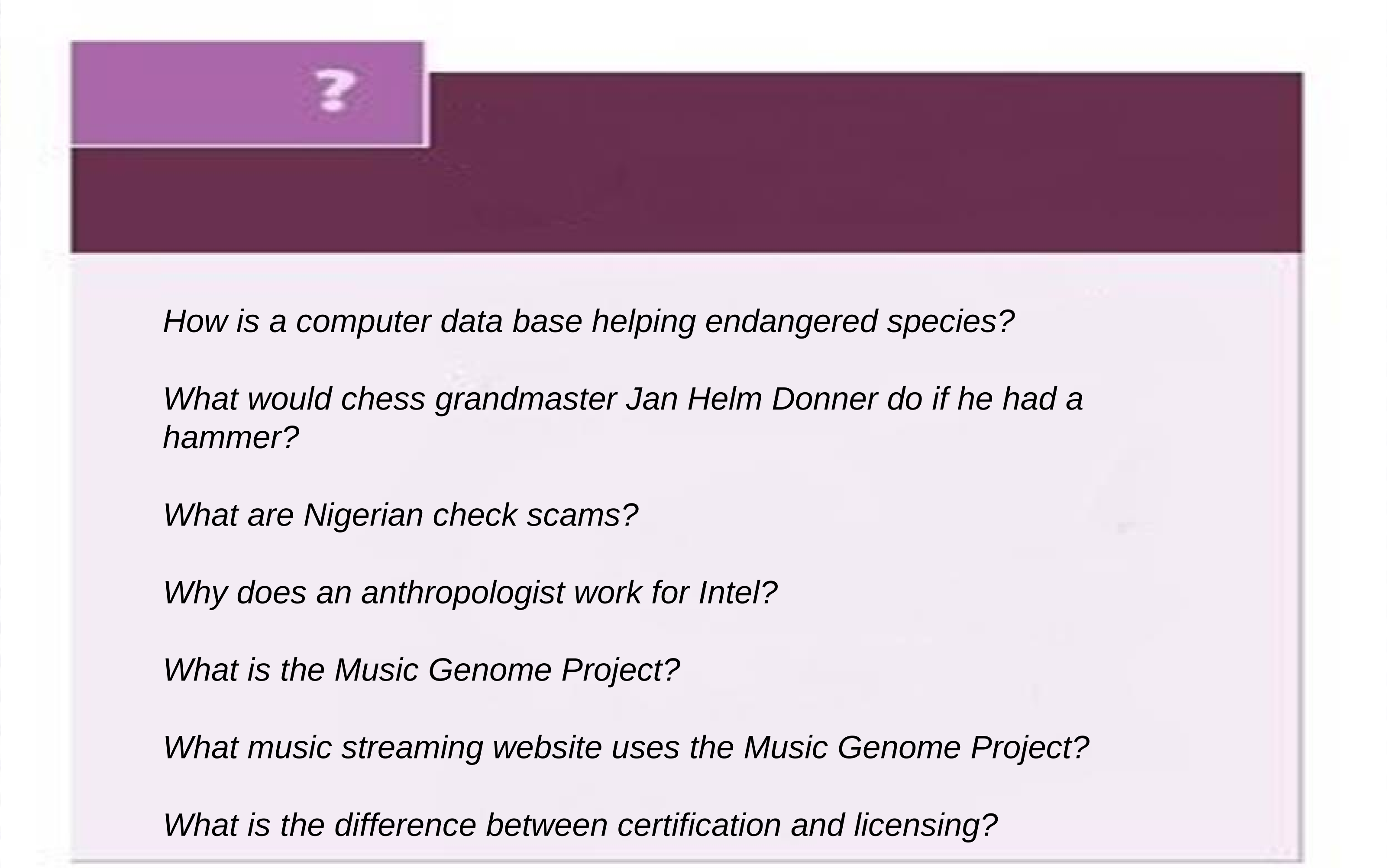
Who am I?



© Karl Staedele/dpa/Corbis

Turing, Atanasoff, Eckert, and Mauchly were my contemporaries. Why were we unaware of each other's work?

Do you know?



How is a computer data base helping endangered species?

What would chess grandmaster Jan Helm Donner do if he had a hammer?

What are Nigerian check scams?

Why does an anthropologist work for Intel?

What is the Music Genome Project?

What music streaming website uses the Music Genome Project?

What is the difference between certification and licensing?