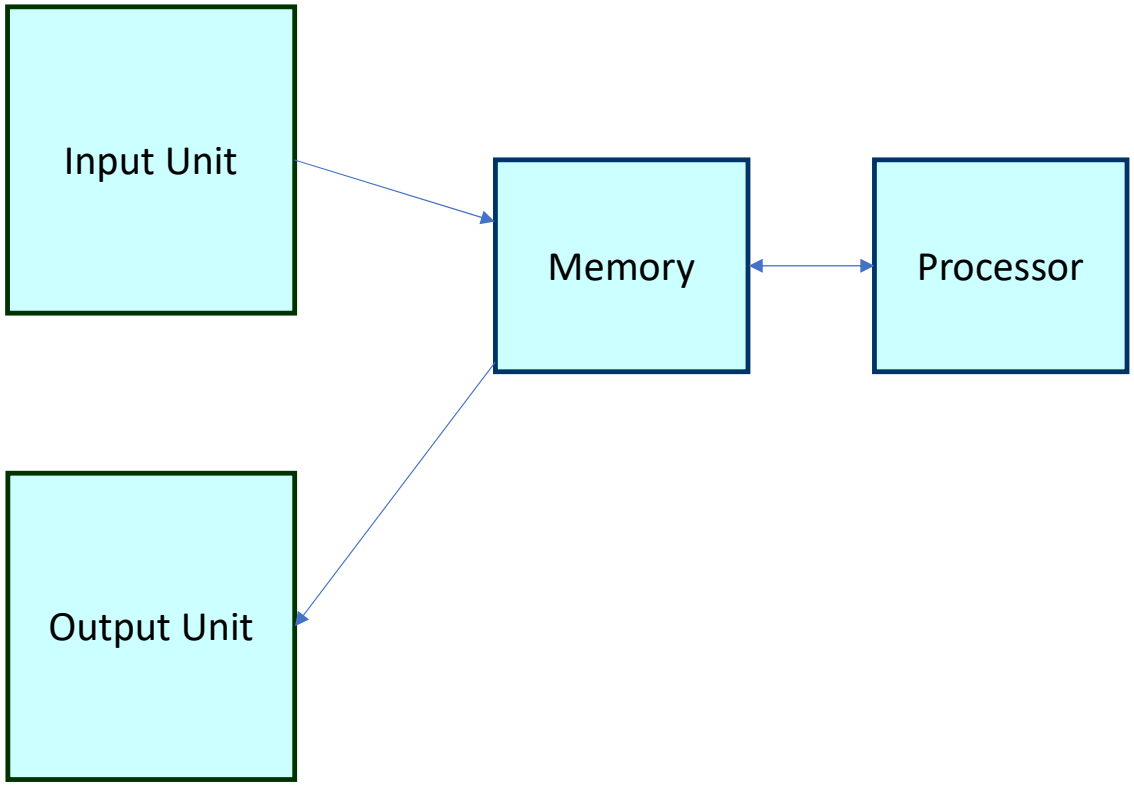
# Computer Architecture

# Computer System



# Memory

- Memory unit stores instructions and data.
  - Recall, data is represented as a series of bits.
  - To store data, memory unit thus stores bits.
- Processor reads instructions and reads/writes data from/to the memory during the execution of a program.
  - In theory, instructions and data could be fetched one bit at a time.
  - In practice, a group of bits is fetched at a time.
  - Group of bits stored or retrieved at a time is termed as "word"
  - Number of bits in a word is termed as the "word length" of a computer.
- In order to read/write to and from memory, a processor should know where to look:
  - "Address" is associated with each word location.

# Memory

- Primary storage of the computer consists of RAM units.
  - Fastest, smallest unit is Cache.
  - Slowest, largest unit is Main Memory.
- Store large amounts of data on secondary storage devices:
  - Solid State drives
  - Optical disks (CD-ROMS).
  - Magnetic disks and tapes,
  - Online - cloud
  - Access to the data stored in secondary storage in slower, but take advantage of the fact that some information may be accessed infrequently.
- Cost of a memory unit depends on its access time, lesser access time implies higher cost.

# Memory

**Main Memory**

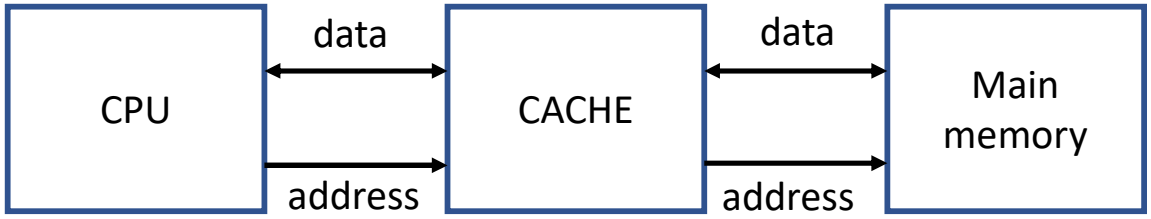
RAM – Random access memory
volatile memory that stores information temporarily.

ROM – Read Only Memory
non-volatile memory; retains data/instruction after power down
EPROMS these can be rewritten

**Cache memory**

Organization of cache and main memory

Cache memory can store both data and instructions.
Both data cache and instructions cache increase the performance of a processor.




Cache :faster access to frequently used data

# ALU

- Operations are executed in the Arithmetic and Logic Unit (ALU).
  - Arithmetic operations such as addition, subtraction.
  - Logic operations such as comparison of numbers.
- In order to execute an instruction, operands (values) need to be brought into the ALU from the memory.
  - Operands are stored in general purpose registers available in the ALU.
  - Access times of general purpose registers are faster than the cache.
- Results of the operations are stored back in the memory or retained in the processor for immediate use.

# Control Unit

- Operation of a computer can be summarized as:
  - Accepts information from the input units (Input unit).
  - Stores the information (Memory).
  - Processes the information (ALU).
  - Provides processed results through the output units (Output unit).
- Operations of Input unit, Memory, ALU and Output unit are coordinated by Control unit.
- Instructions control “what” operations take place (e.g. data transfer, processing).
- Control unit generates timing signals which determines “when” a particular operation takes place.

# CPU

- **Program counter:**
  - Is a register in a computer processor that contains the address (location) of the instruction being executed at the current time. As each instruction gets fetched, the program counter increases its stored value by the length of current instruction

- **Address Decoder:**
  - Is a binary decoder that has two or more inputs for address bits and one or more outputs for device selection signals. When the address for a particular device appears on the address inputs, the decoder asserts the selection output for that device.

- **Instruction Decoder:**
  - The instruction decoder is the circuit that decodes an opcode. Then activates different part of the processor which effectively services the machine
  - Rem. FSM where we are in one state and we have an input and based on this, this will activate different parts of the processor e.g. ALU etc..

# CPU

**File registers:**

Are a memory locations inside the processor that can be written to, read from or both. They have addresses which we use to access them and some of which have labels

**Accumulator:**

is a register for short-term, intermediate storage of arithmetic and logic data in a computer's CPU (central processing unit).

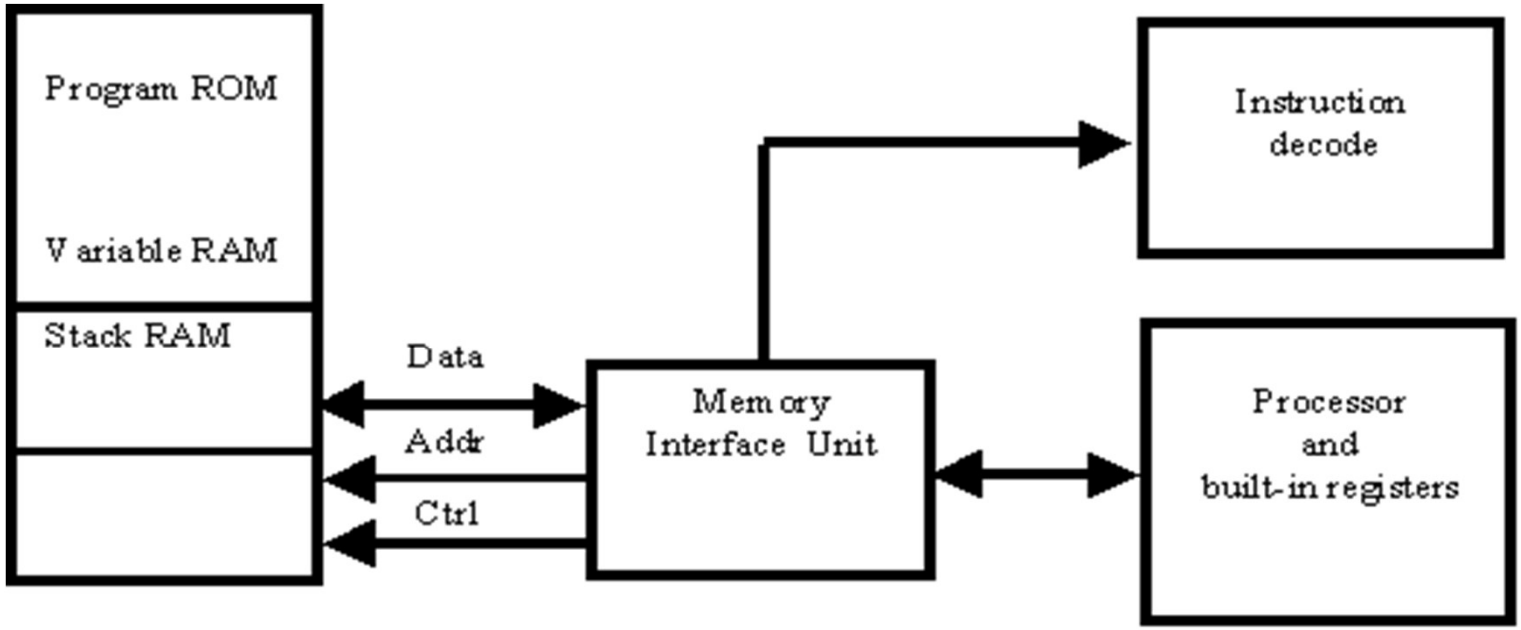
**ALU Arithmetic Logic Unit:**

is the digital circuitry used to perform arithmetic and logic operations.

# Princeton Vs Harvard

The **memory interface unit is responsible for arbitrating access to the memory space** between reading instructions (based upon the current program counter) and passing data back and forth with the processor and its internal registers.

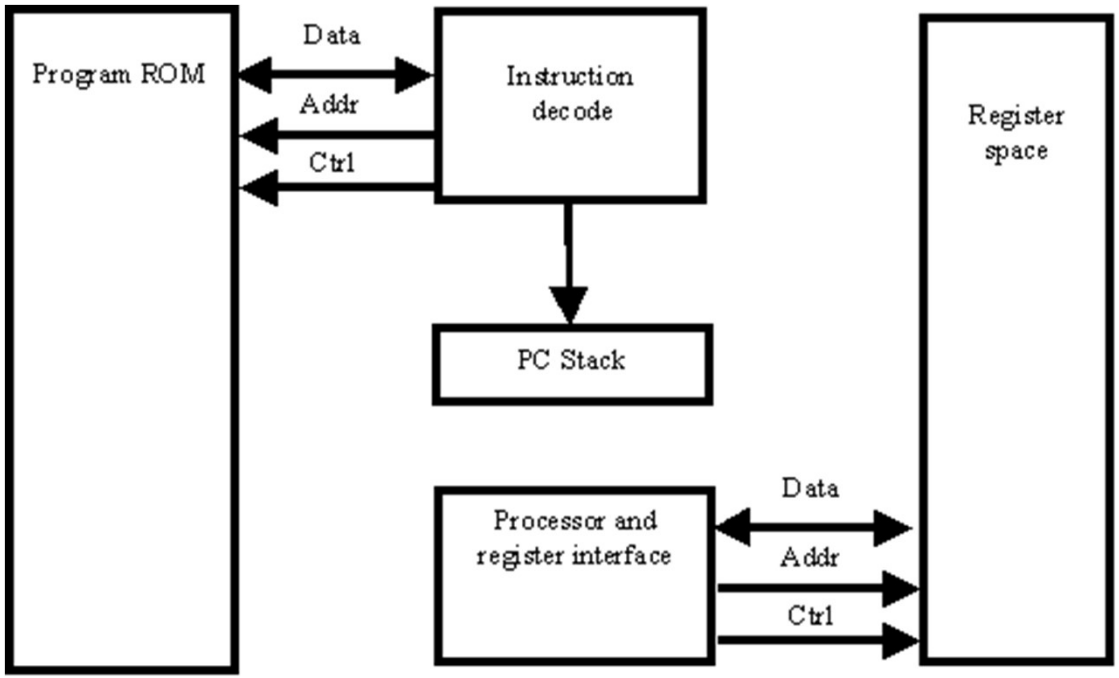
.




So we use the same memory and data path for programs, and data

# Princeton Vs Harvard

The Harvard architecture was largely ignored until the late 1970s when **microcontroller** manufacturers realized that the architecture had advantages for the devices they were currently designing



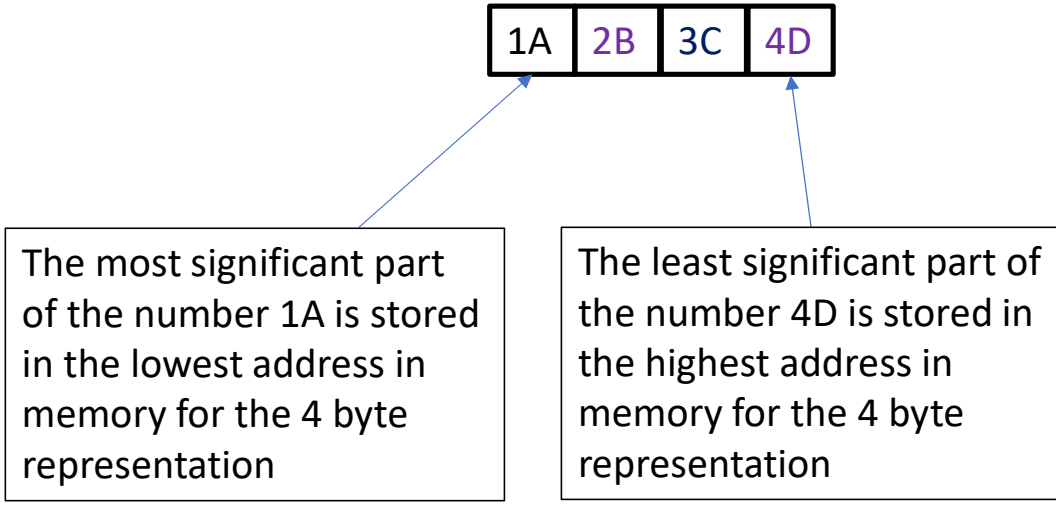

So we use the different memories and data paths for programs, and data memory

# Big Endian/ Little Endian

- Machines store values in two different orders
- 1. Big Endian – The Most significant Byte is stored on left hand side
- 2. Little Endian – The Least Significant Byte is stored on the right hand side
- We in every day use, use Big endian, for example 673 where the 6 represents 6 hundred (the most significant digit is on the left hand side)
- Intel and AMD x86-64 use little endian, IP/TCP uses Big Endian

# Big Endian/ Little Endian

- Consider hex value 0x 1A2B3C4D
- On a computer this is a 4 byte number and
- In Big Endian its is stored as



# Big Endian/ Little Endian

- Consider hex value 0x 1A2B3C4D
- On a computer this is a 4 byte number and
- In little Endian its is stored as


The most significant part of the number 1A is stored in the lowest address in memory for the 4 byte representation

The least significant part of the number 4D is stored in the highest address in memory for the 4 byte representation

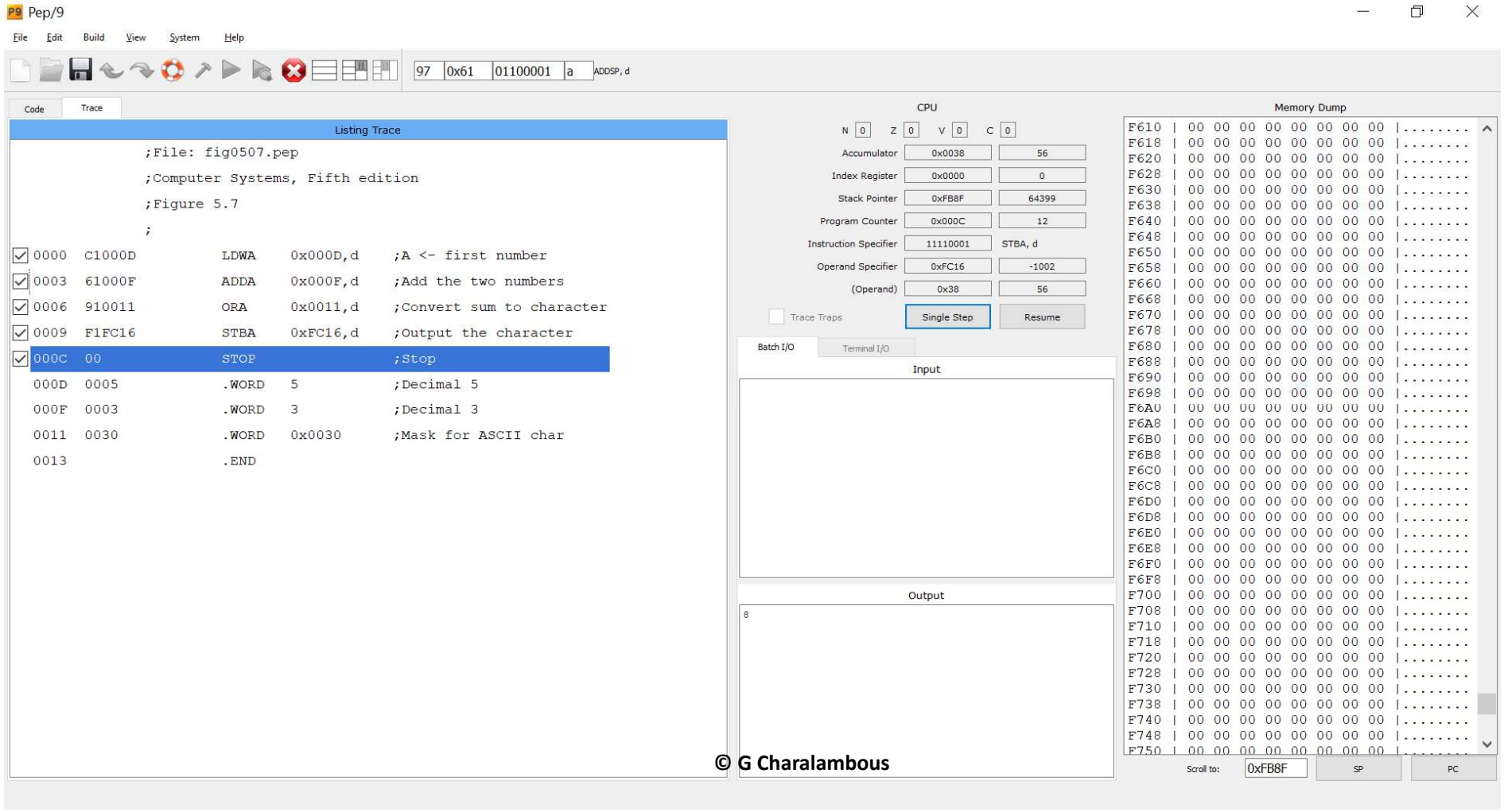
# PEP

- This is an application that simulates the way assembly code gets executed on a CPU
- If we wish to program a CPU we need to consider its instruction set
- The CPU is an example of a Finite State Machine
- Consider it as a mechanism driven by a clock. At a given time it is in a given state we control the mechanism by giving instructions based on these it will at the next time interval move to the next state and so on
- The circuitry of the mechanism is based on logic where we define TRUE with a high voltage (+5v) and False with a low voltage(0v)

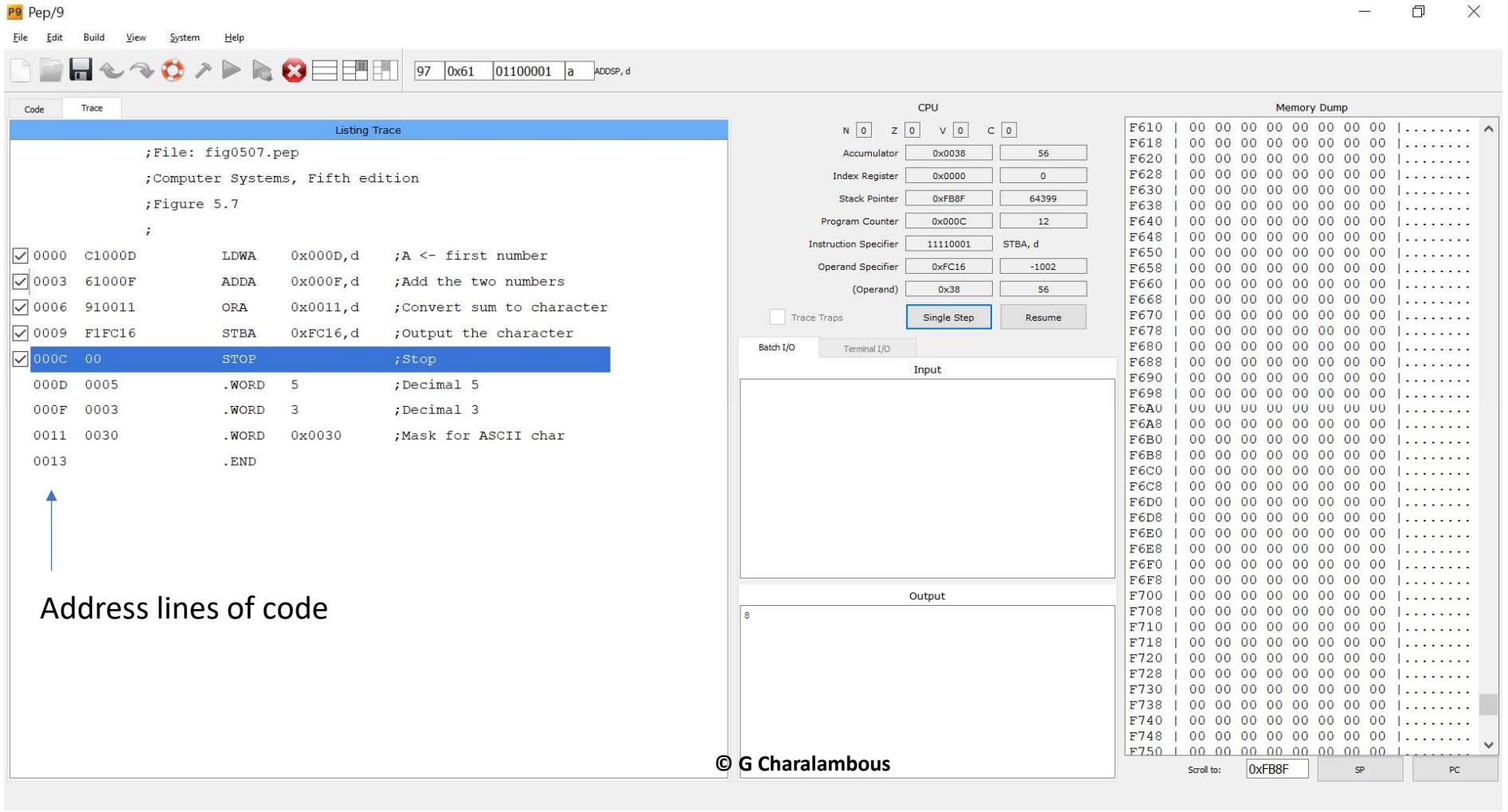
# PEP

- Consider a switch we have two states on or off
- In electronics we use transistors – you can consider this a switch
- In modern CPUs we use transistors and build all the logic
- for the 8086 CPU it had a clock speed of 10MHz that's 10 million pulses every second and had 29000 transistors; produced in 1976; a 16 bit machine.
- The i7 core runs around 4 GHz ( 4 000,000,000 pulses every second) with 3 billion transistors; produced 2008; a 64 bit machine.

# PEP screen shot

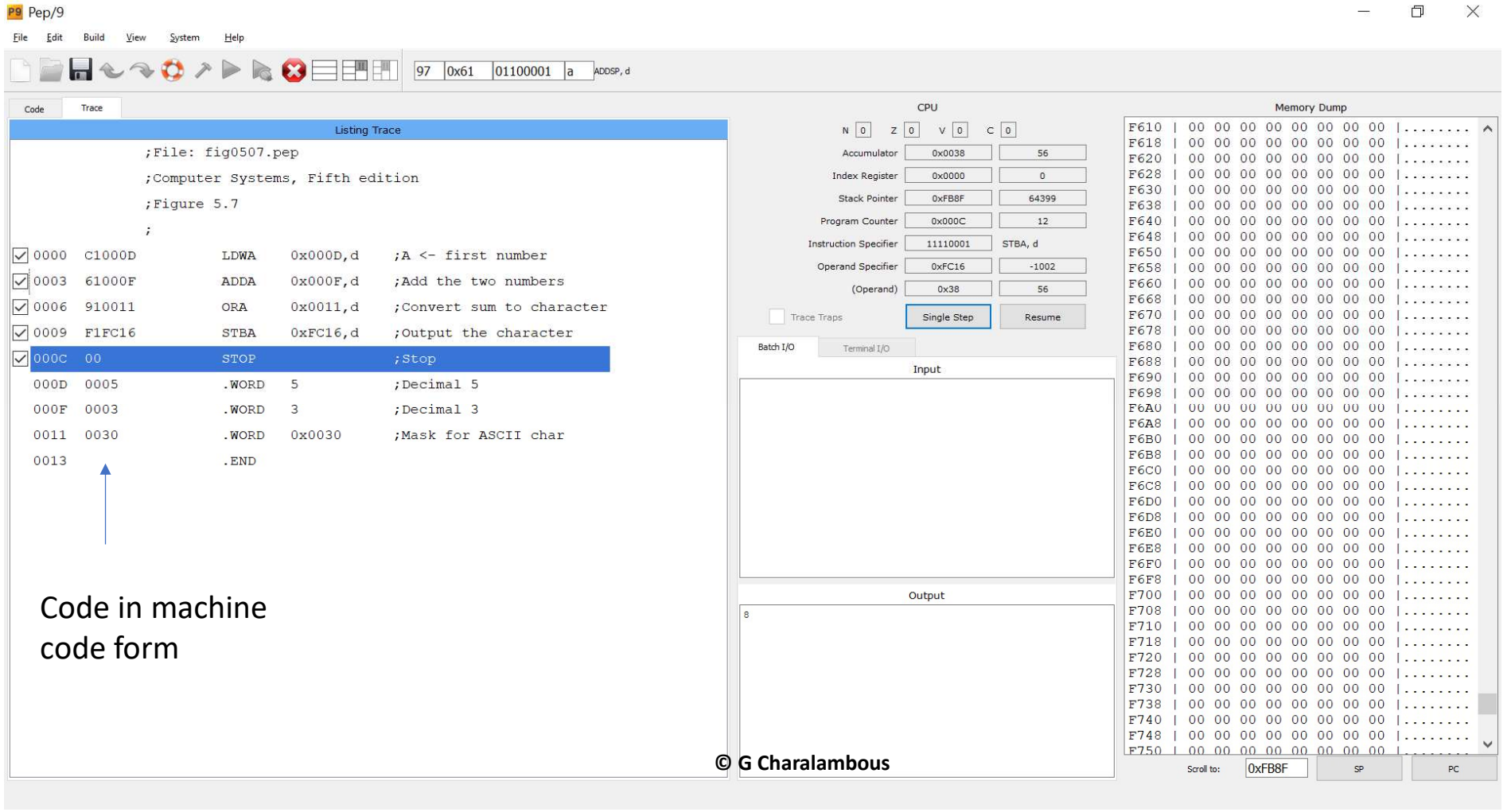


# PEP screen shot



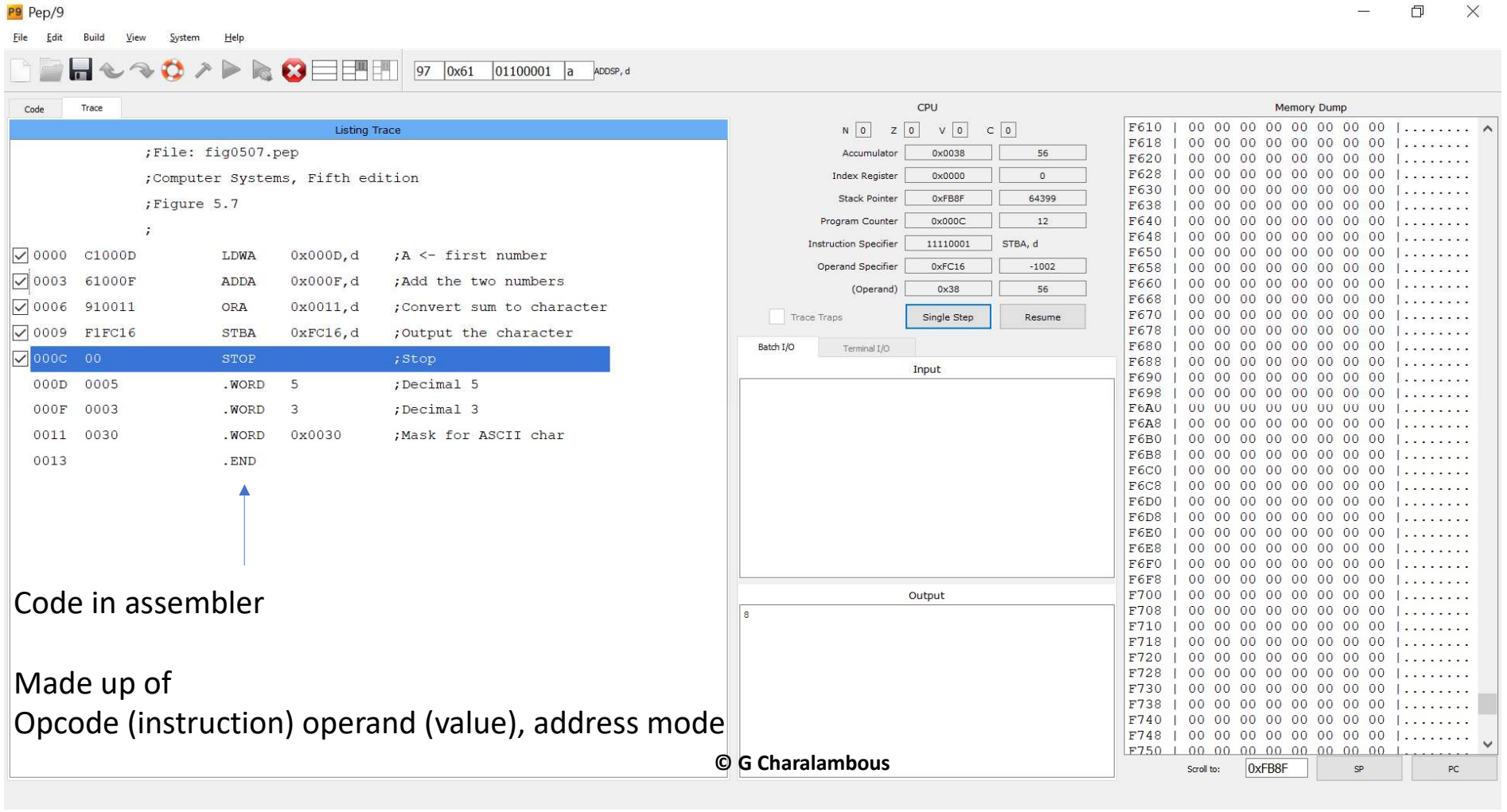

Address lines of code

# PEP screen shot




Code in machine code form

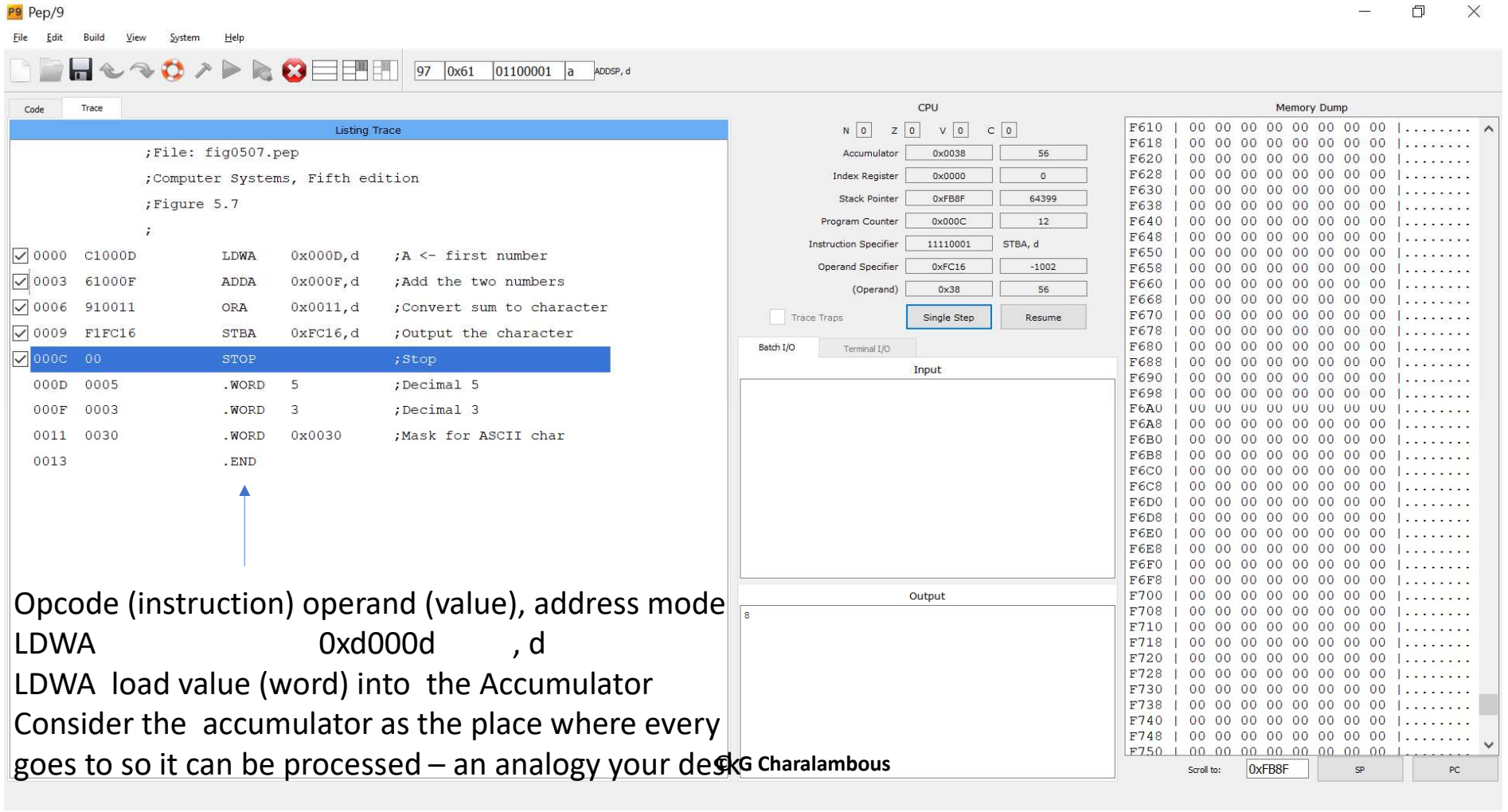
# PEP screen shot




Code in assembler

Made up of
Opcode (instruction) operand (value), address mode

# PEP screen shot



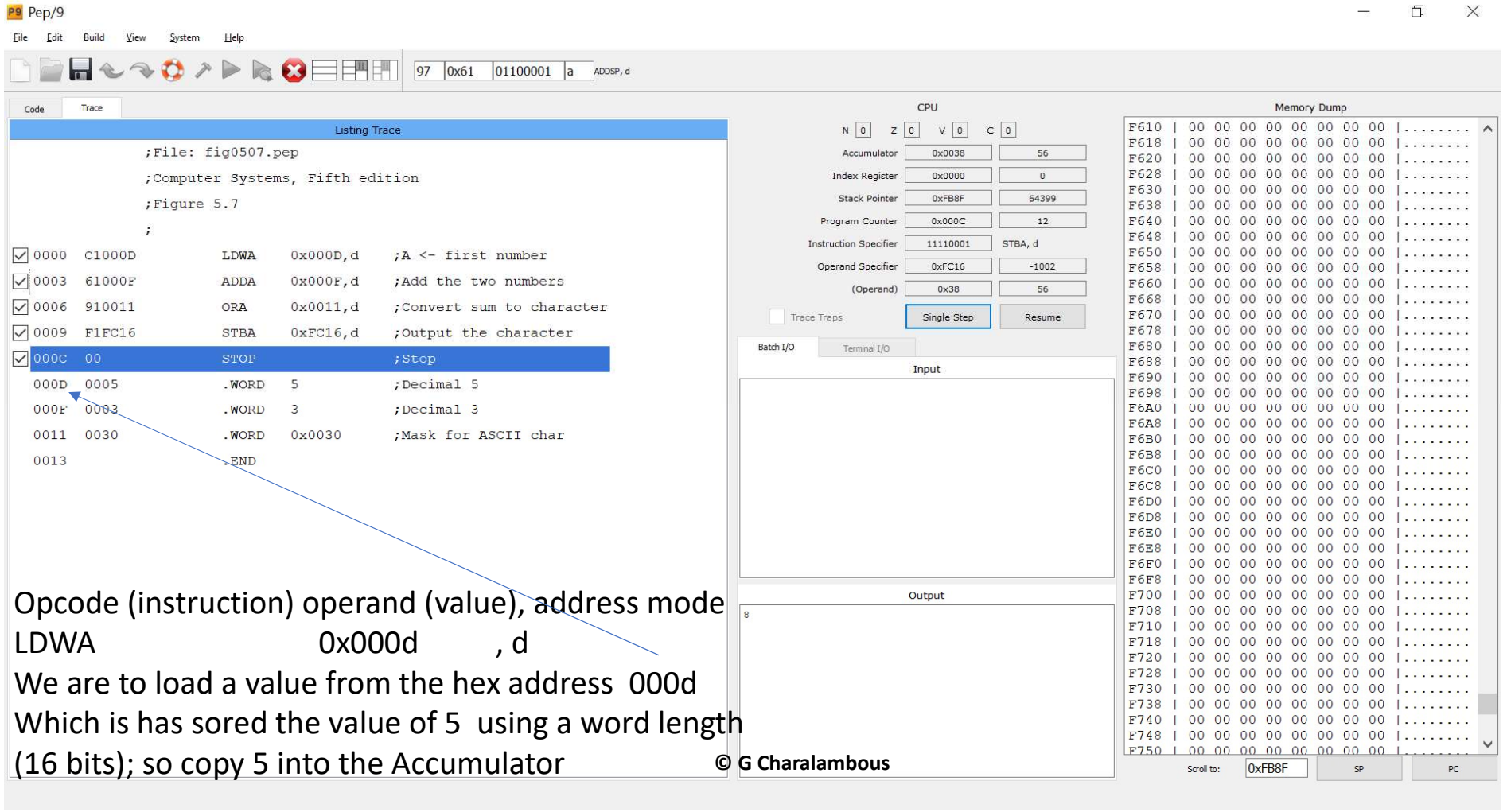

Opcode (instruction) operand (value), address mode

LDWA 0xd000d , d

LDWA load value (word) into the Accumulator

Consider the accumulator as the place where every goes to so it can be processed – an analogy your desk

# PEP screen shot



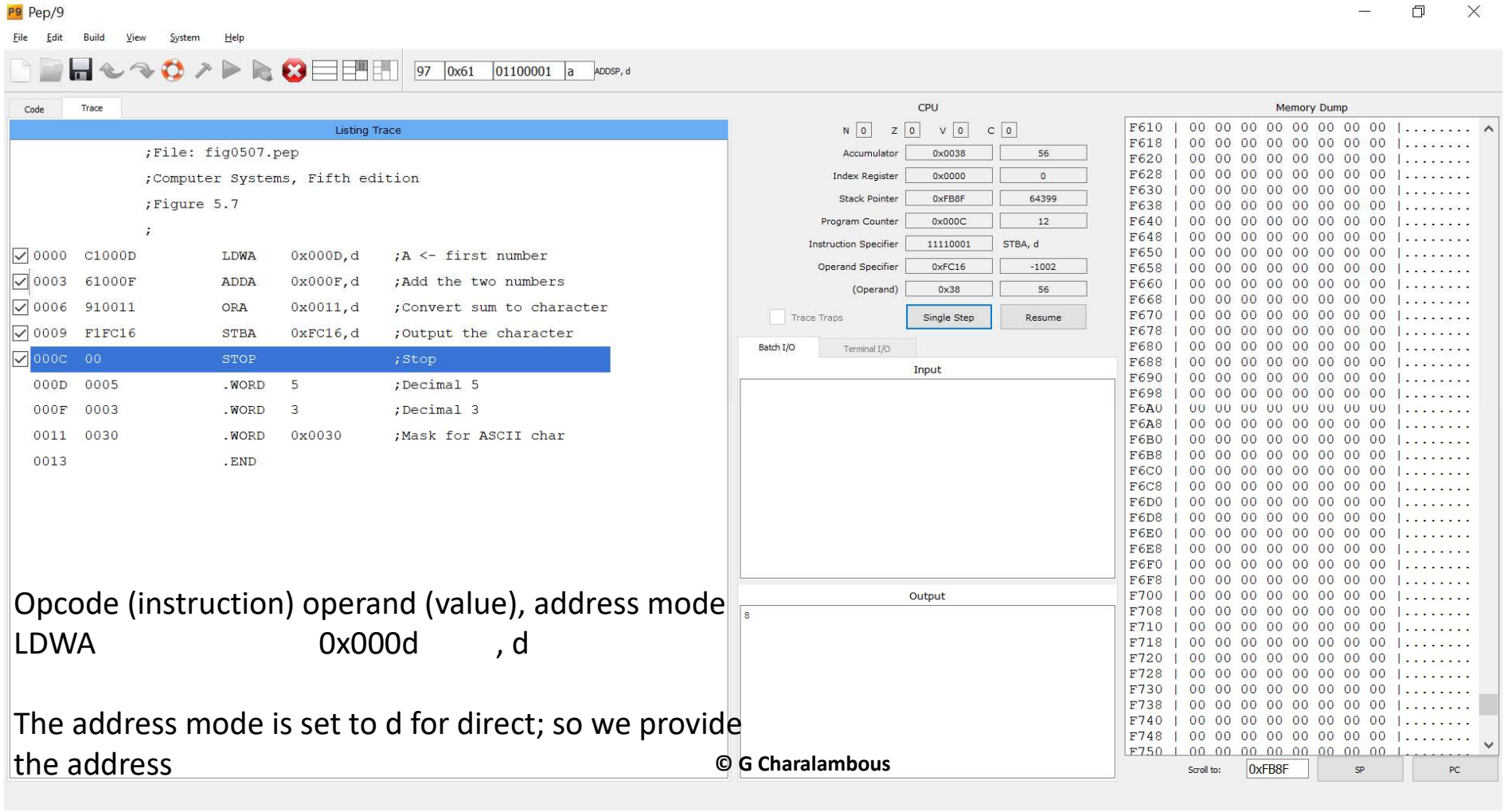

Opcode (instruction) operand (value), address mode

LDWA 0x000d , d

We are to load a value from the hex address 000d

Which is has sored the value of 5 using a word length (16 bits); so copy 5 into the Accumulator

# PEP screen shot

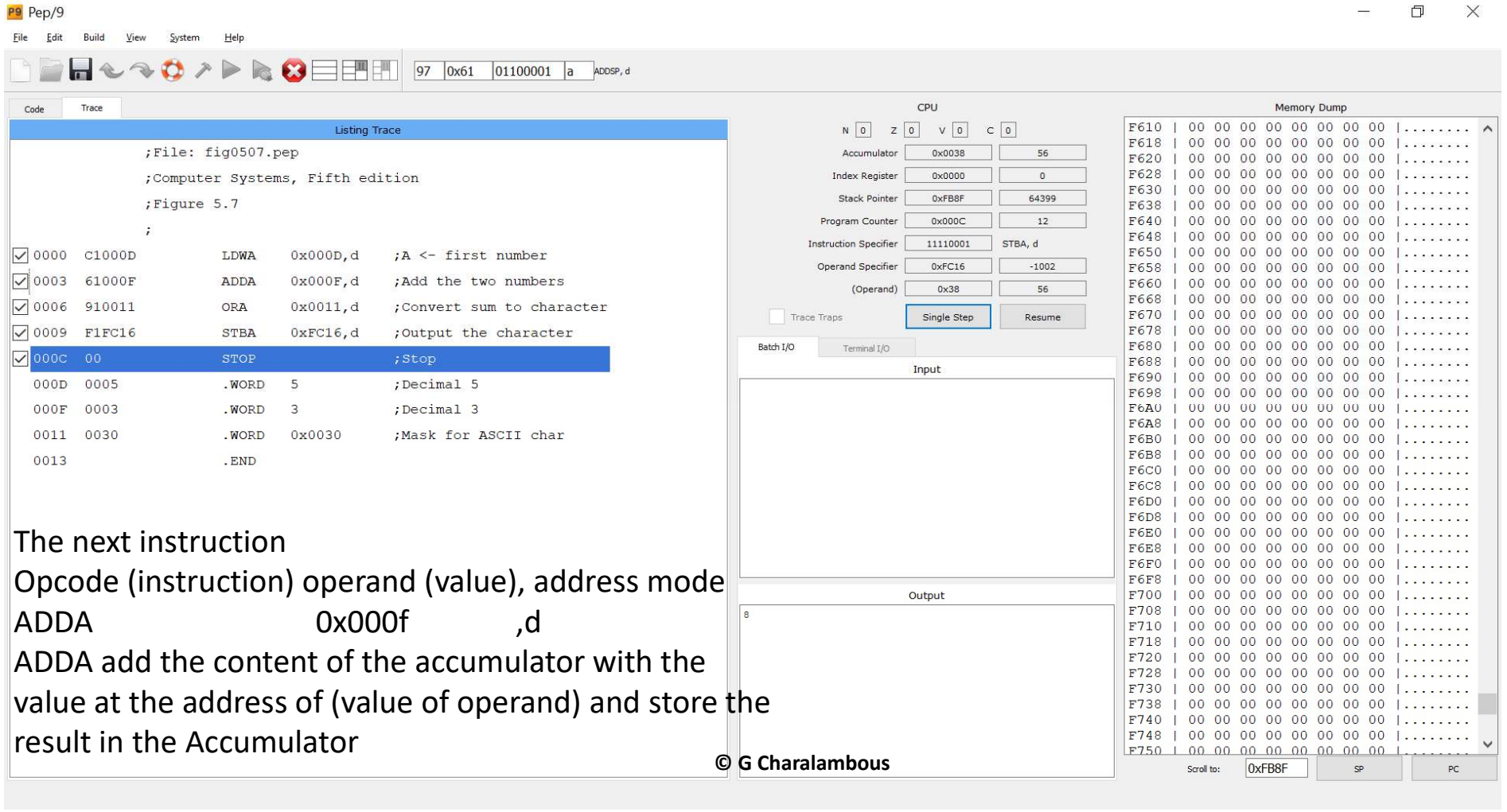



Opcode (instruction) operand (value), address mode

LDWA 0x000d , d

The address mode is set to d for direct; so we provide the address

# PEP screen shot



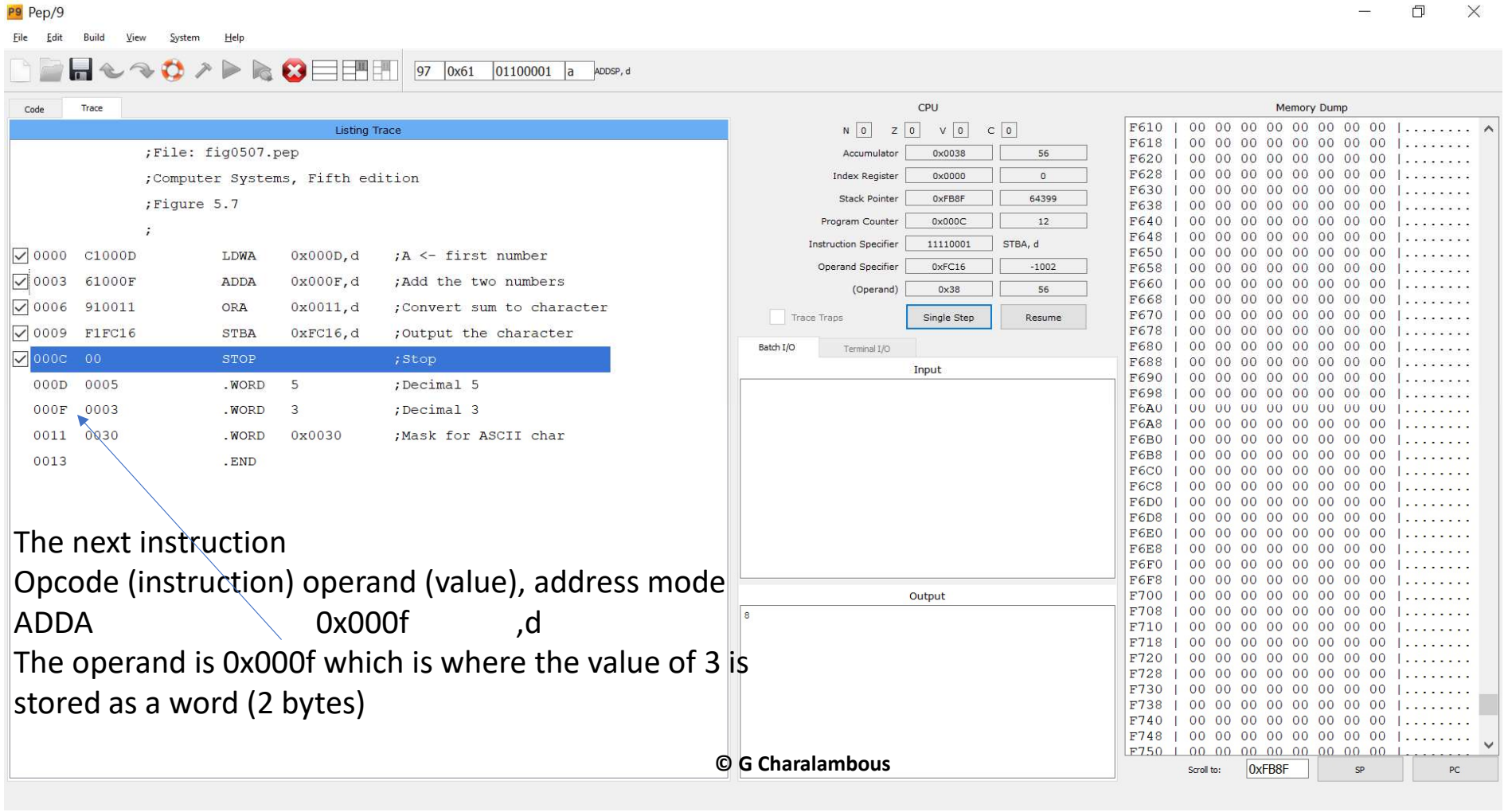

The next instruction

Opcode (instruction) operand (value), address mode

ADDA 0x000f ,d

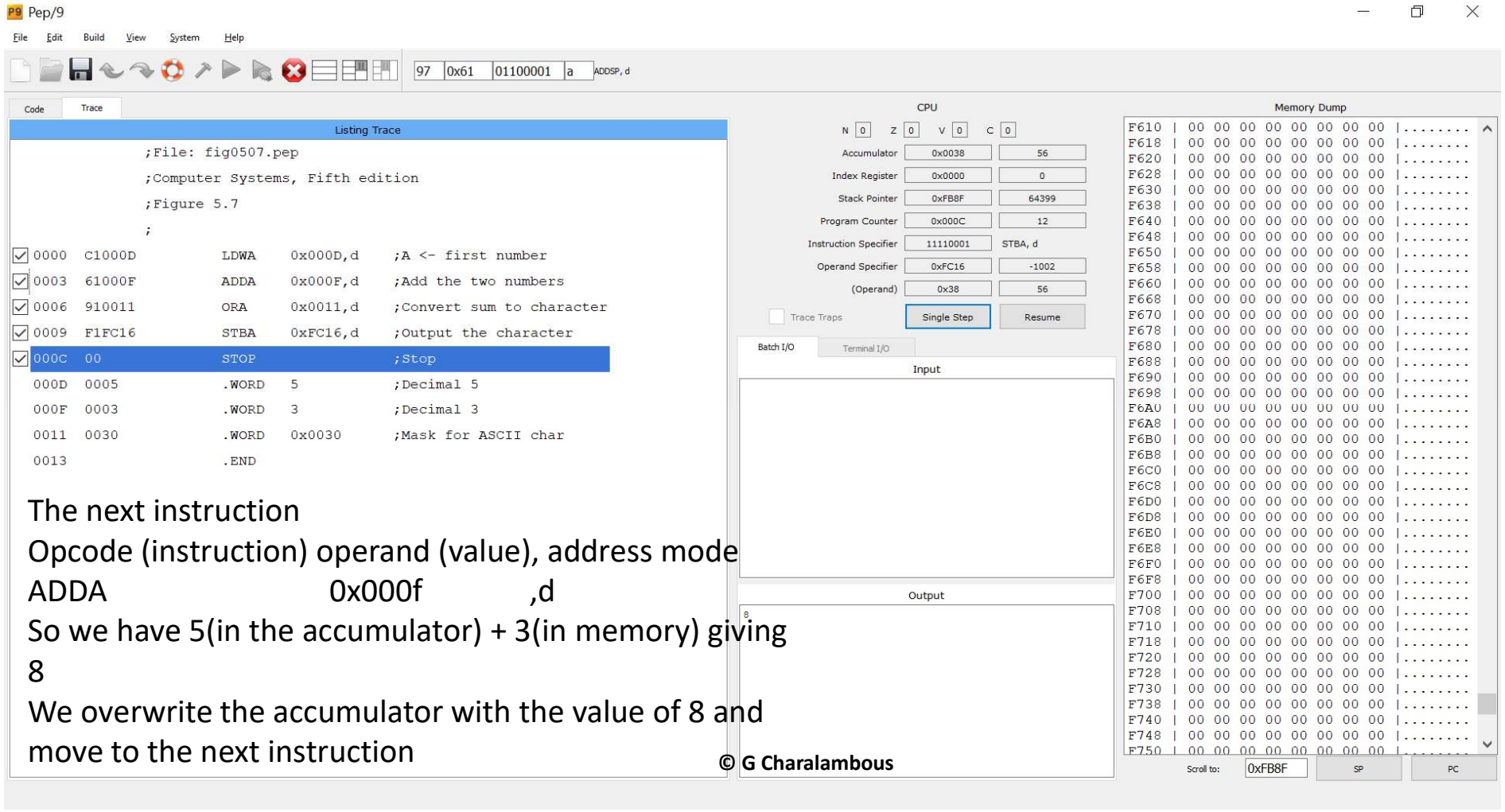
ADDA add the content of the accumulator with the value at the address of (value of operand) and store the result in the Accumulator

# PEP screen shot




The next instruction
Opcode (instruction) operand (value), address mode
ADDA 0x000f ,d
The operand is 0x000f which is where the value of 3 is stored as a word (2 bytes)

# PEP screen shot




The next instruction

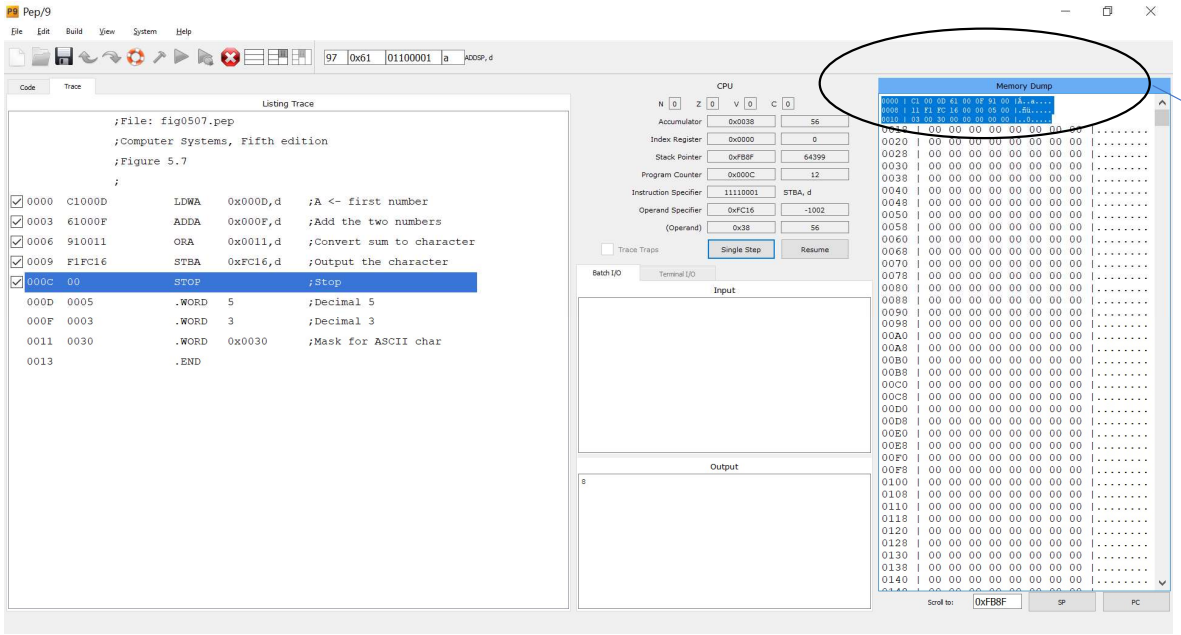
Opcode (instruction) operand (value), address mode

ADDA 0x000f ,d

So we have 5(in the accumulator) + 3(in memory) giving 8

We overwrite the accumulator with the value of 8 and move to the next instruction

# PEP




0000 | C1 00 0D 61 00 0F 91 00 |Á..a....
0008 | 11 F1 FC 16 00 05 00 |.ñü.....
0010 | 03 00 30 00 00 00 00 00 |..0.....

These are the machine code instructions
Each 2 hex digits represents a byte in memory

At address 000D we store the value of 5