

ADVANCED MICROPROCESSOR

PROJECT REPORT

INTERPRETER

SUBMITTED BY:
ABHINAV MATHAROO (13105021)
PRATEEK GAMBHIR (13105066)
NIKHILESH GOEL (13105082)

INTRODUCTION

The **8086** is a 16-bit microprocessor chip designed by Intel between early 1976 and mid-1978, when it was released. The Intel 8088, released in 1979, was a slightly modified chip with an external 8-bit data bus (allowing the use of cheaper and fewer supporting ICs), and is notable as the processor used in the original IBM PC design, including the widespread version called IBM PC XT. The 8086 gave rise to the x86 architecture which eventually became Intel's most successful line of processors.

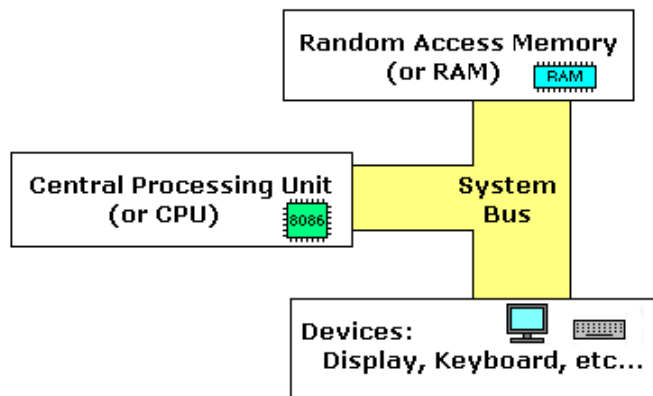
ASSEMBLY LANGUAGE

An assembly language (or assembler language) is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions. Each assembly language is specific to a particular computer architecture, in contrast to most high-level programming languages, which are generally portable across multiple architectures, but require interpreting or compiling.

Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

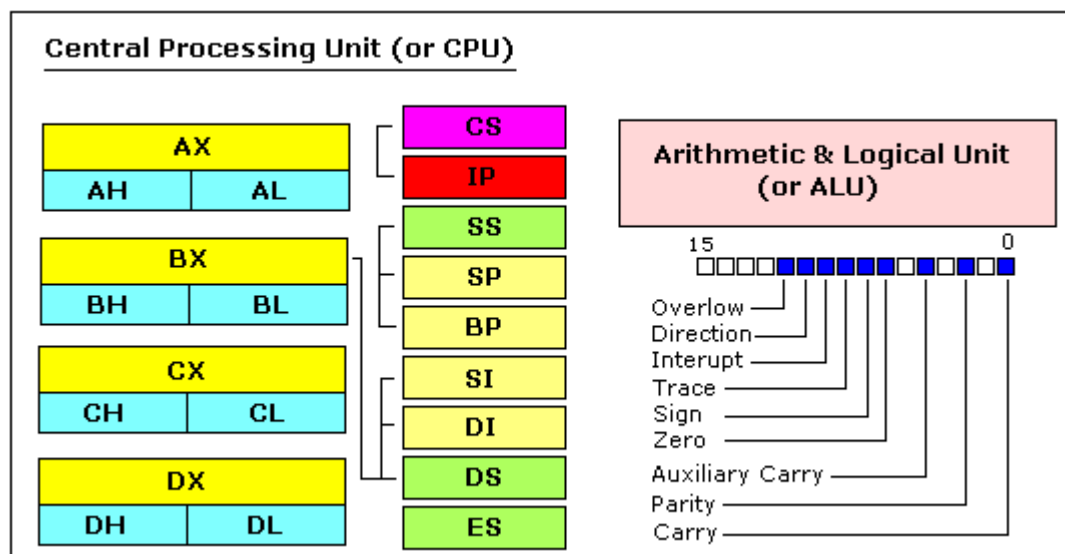
Assembly language uses a mnemonic to represent each low-level machine instruction or operation. Typical operations require one or more operands in order to form a complete instruction, and most assemblers can therefore take labels, symbols and expressions as operands to represent addresses and other constants, freeing the programmer from tedious manual calculations. Macro assemblers include a macroinstruction facility so that (parameterized) assembly language text can be represented by a name, and that name can be used to insert the expanded text into other code. Many assemblers offer additional mechanisms to facilitate program development, to control the assembly process, and to aid debugging.

Assembly language is a low level programming language. You need to get some knowledge about computer structure in order to understand anything. The simple computer model as I see it:



The system bus (shown in yellow) connects the various components of a computer. The CPU is the heart of the computer, most of computations occur inside the CPU. RAM is a place to where the programs are loaded in order to be executed.

INSIDE THE CPU



INTERPRETER

In computer science, an interpreter is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without previously compiling them into a machine language program. An interpreter generally uses one of the following strategies for program execution:

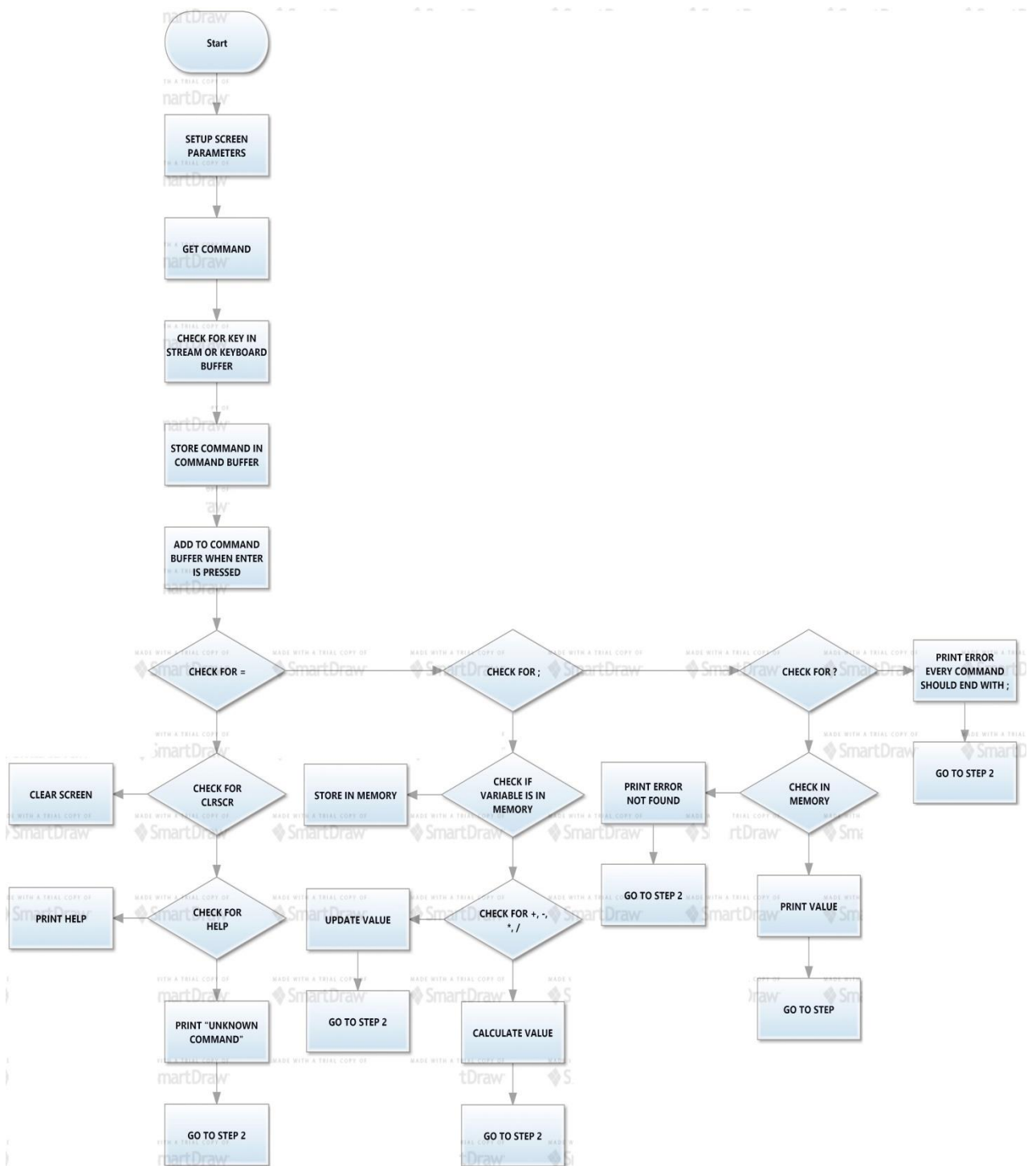
1. parse the source code and perform its behaviour directly.
2. translate source code into some efficient intermediate representation and immediately execute this.

3. explicitly execute stored precompiled code^[1] made by a compiler which is part of the interpreter system.

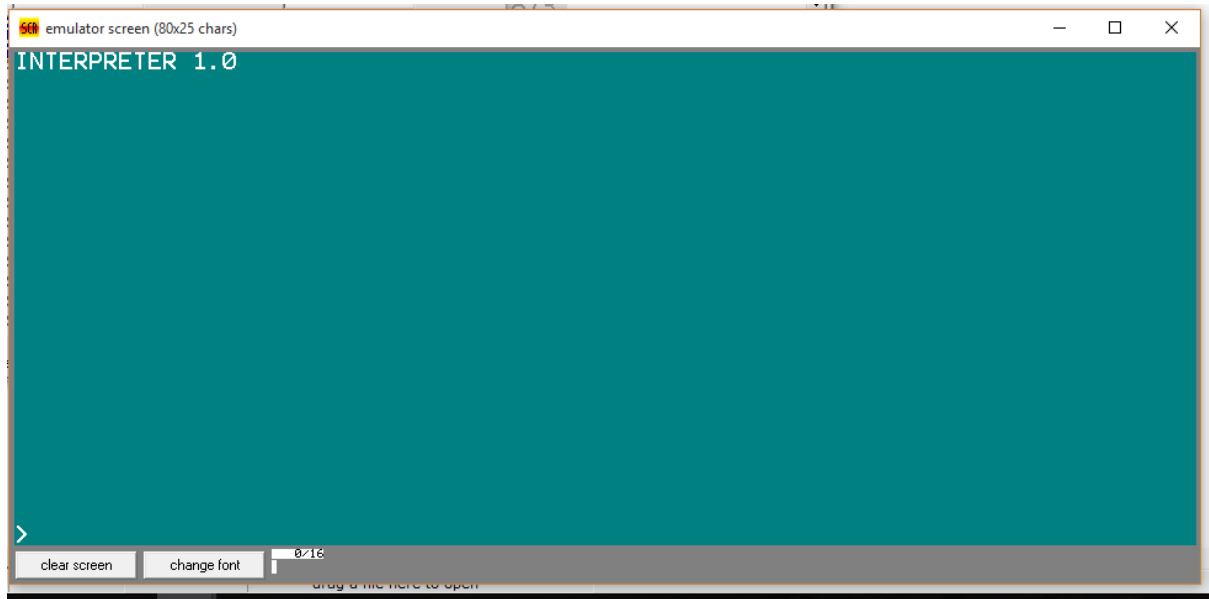
Early versions of Lisp programming language and Dartmouth BASIC would be examples of the first type. Perl, Python, MATLAB, and Ruby are examples of the second, while UCSD Pascal is an example of the third type. Source programs are compiled ahead of time and stored as machine independent code, which is then linked at run-time and executed by an interpreter and/or compiler (for JIT systems). Some systems, such as Smalltalk, contemporary versions of BASIC, Java and others may also combine two and three.

While interpretation and compilation are the two main means by which programming languages are implemented, they are not mutually exclusive, as most interpreting systems also perform some translation work, just like compilers. The terms "interpreted language" or "compiled language" signify that the canonical implementation of that language is an interpreter or a compiler, respectively. A high level language is ideally an abstraction independent of particular implementations.

WORKING



The interpreter stores numerical values under the name of a variable and can perform basic arithmetic operations and display the both positive and negative results.

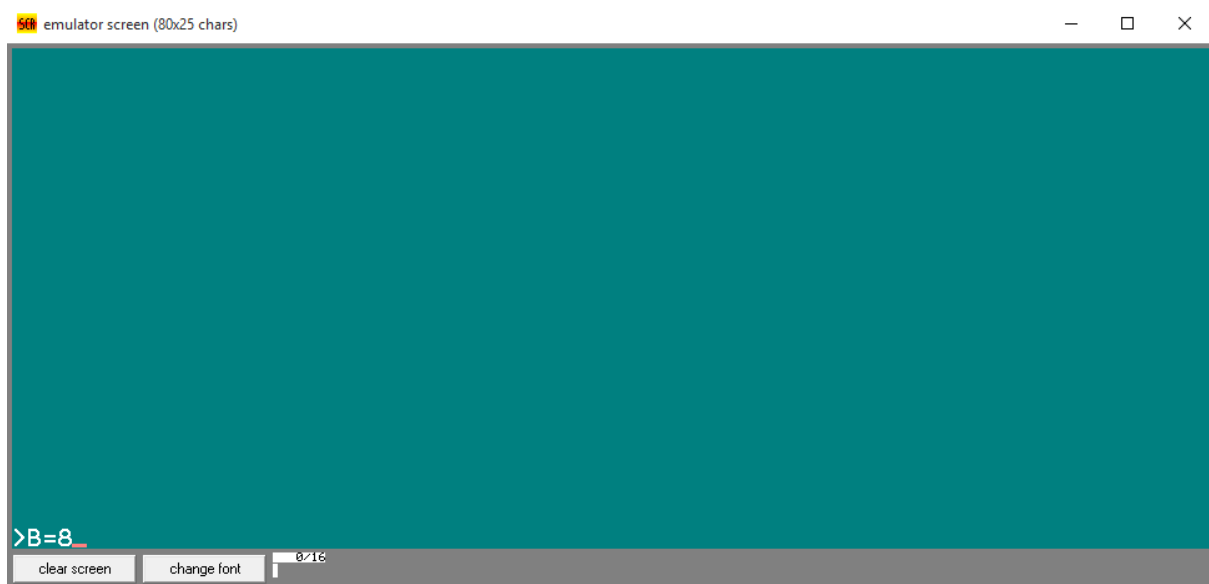


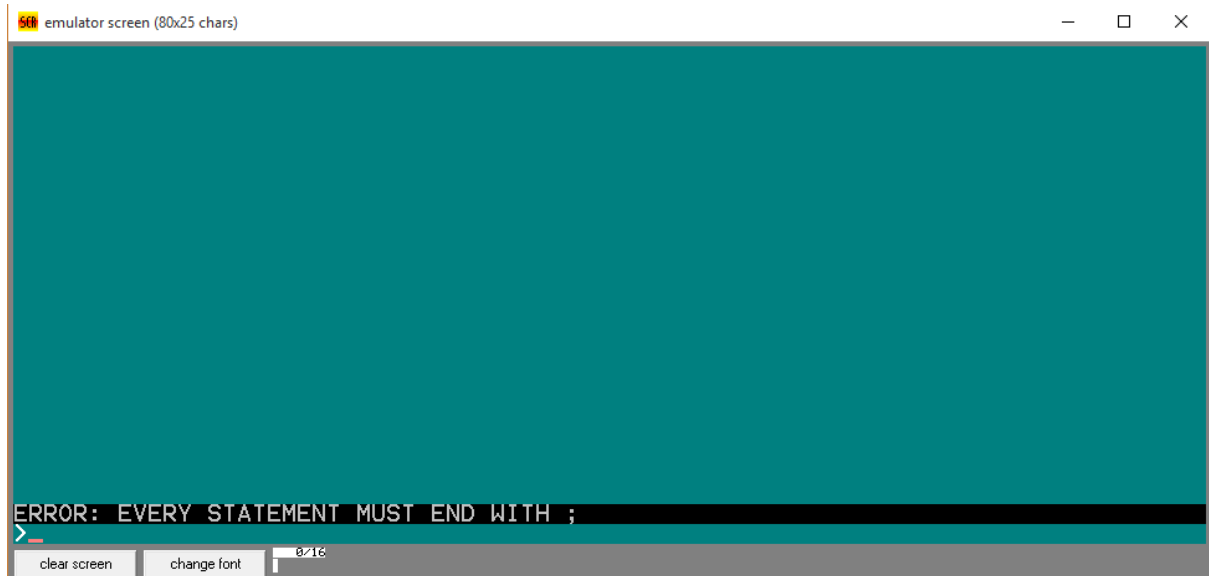
The interpreter accepts commands in the same way as a C compiler. Each command must end with a semi-colon.





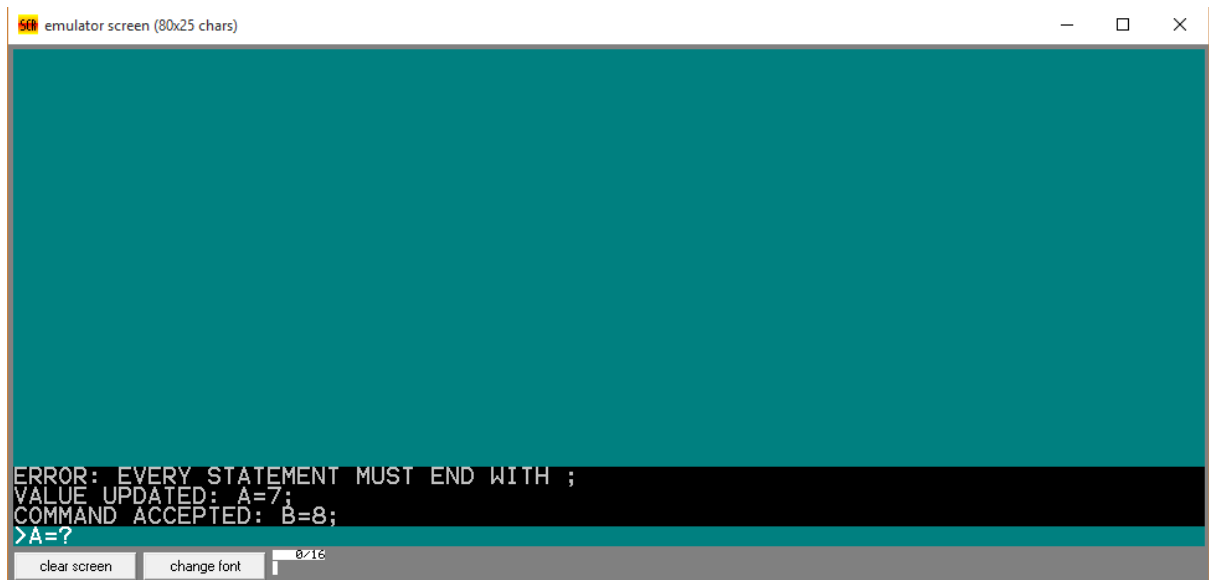
Commands that do not end with a semi colon will not be stored and will generate an error on screen.

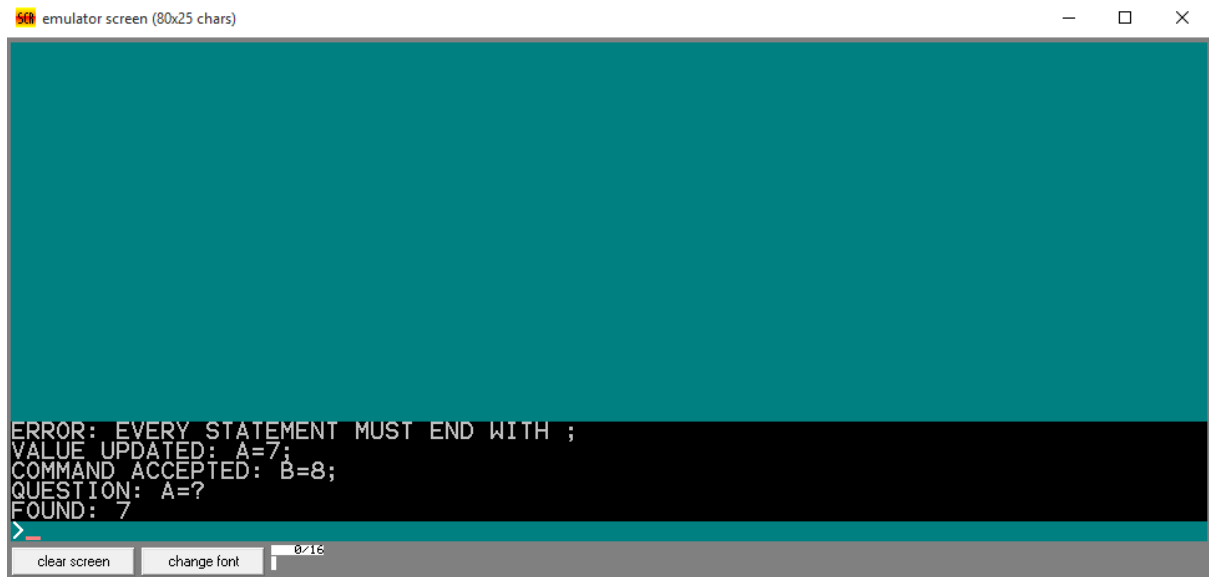




Every variable entered in the correct format will be stored successively at the segment address starting from 4000H. Each variable can have a maximum size of 10 bytes. The variable can have any value between 0-9.

The value of the variable can be checked by entering the variable followed by a question mark. The value of a variable can be updated by re initializing the variable.





emulator screen (80x25 chars)

```
ERROR: EVERY STATEMENT MUST END WITH ;  
VALUE UPDATED: A=7;  
COMMAND ACCEPTED: B=8;  
QUESTION: A=?  
FOUND: 7  
>
```

clear screen change font 0/16

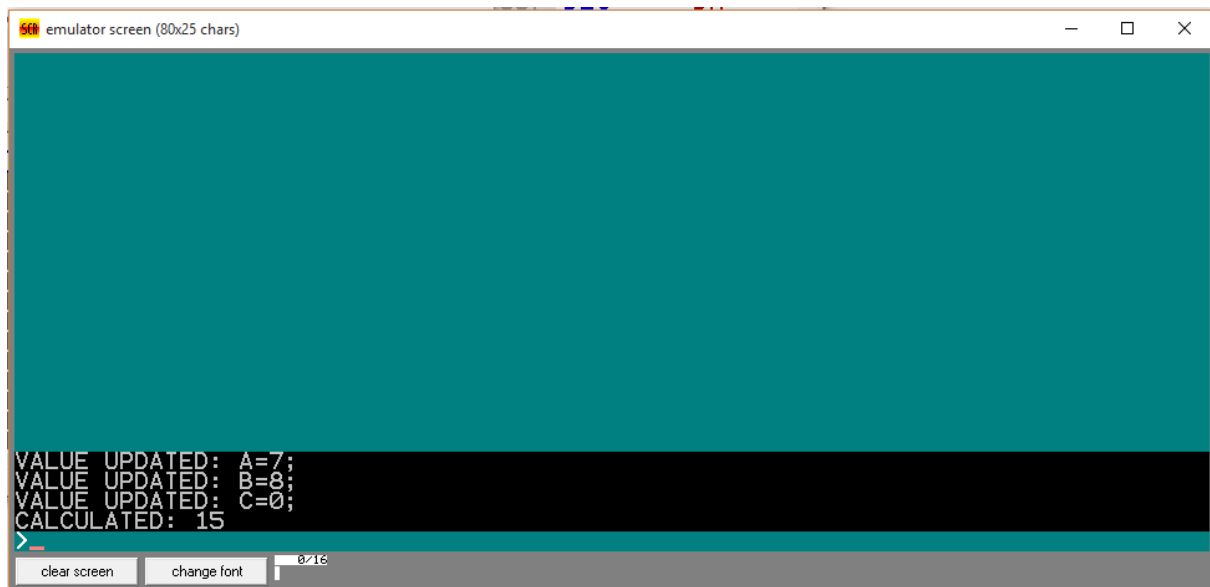
The arithmetic operation can be performed by first initializing a variable to 0 and then writing the operation with the operator in the next command.



emulator screen (80x25 chars)

```
VALUE UPDATED: A=7;  
VALUE UPDATED: B=8;  
>C=A+B;  
-
```

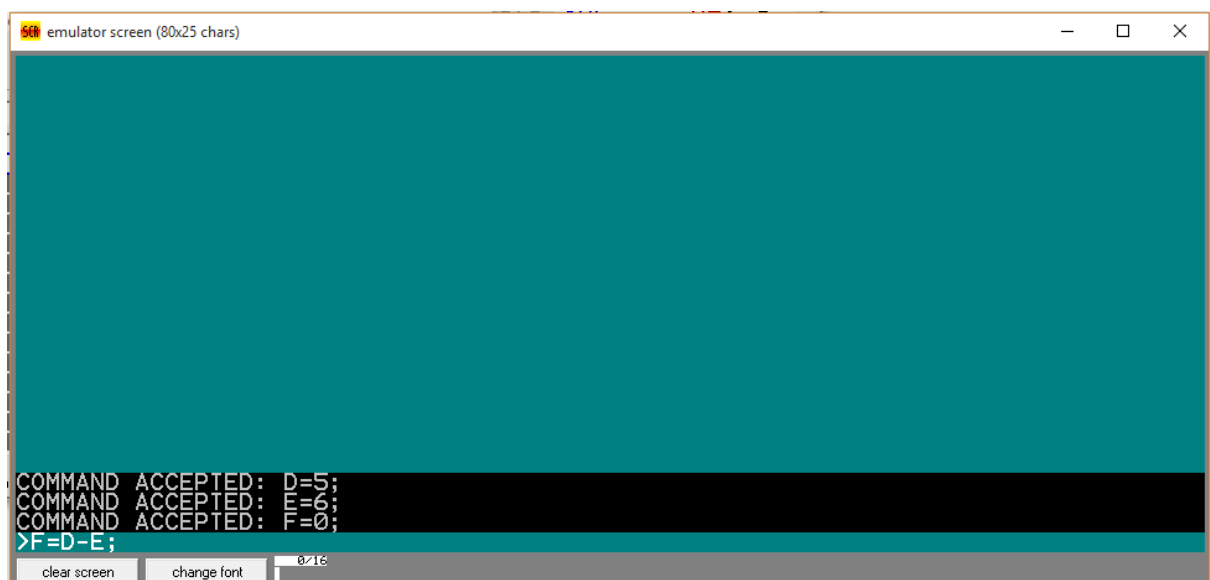
clear screen change font 0/16



emulator screen (80x25 chars)

```
VALUE UPDATED: A=7;  
VALUE UPDATED: B=8;  
VALUE UPDATED: C=0;  
CALCULATED: 15  
>
```

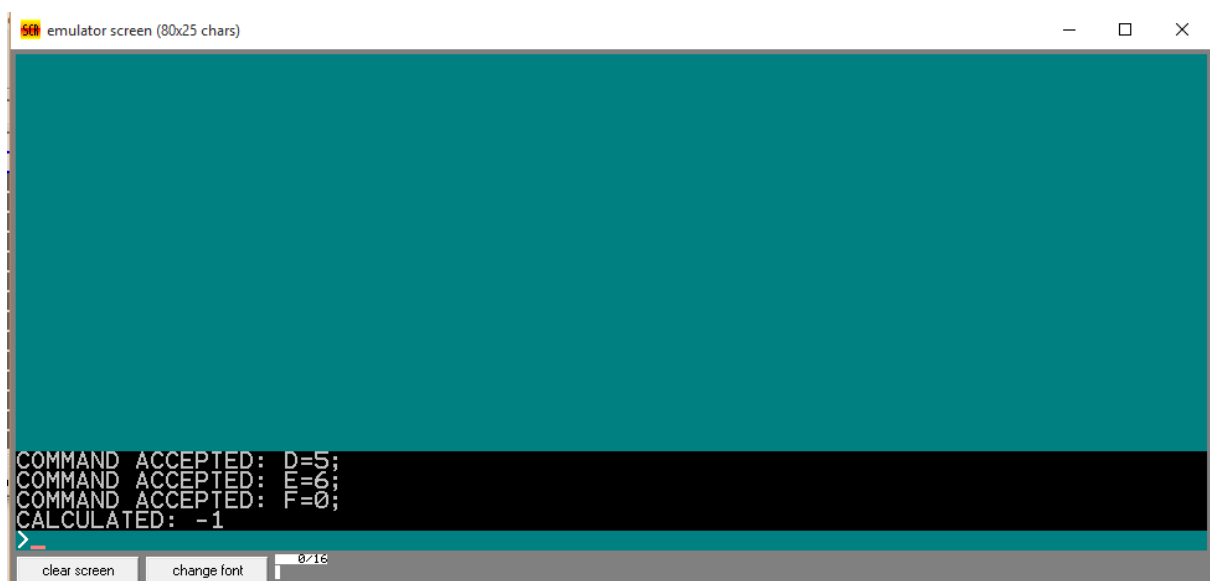
clear screen change font 0/16



emulator screen (80x25 chars)

```
COMMAND ACCEPTED: D=5;  
COMMAND ACCEPTED: E=6;  
COMMAND ACCEPTED: F=0;  
>F=D-E;
```

clear screen change font 0/16



emulator screen (80x25 chars)

```
COMMAND ACCEPTED: D=5;  
COMMAND ACCEPTED: E=6;  
COMMAND ACCEPTED: F=0;  
CALCULATED: -1  
>
```

clear screen change font 0/16

Each of the operations has a specific sub routine in the assembly language code. The compiler jumps to the specific sub routine by comparing the entered command with the ASCII values of the stored symbols. If an unknown symbol is entered no operation is performed.