EXPERIMENT - 8

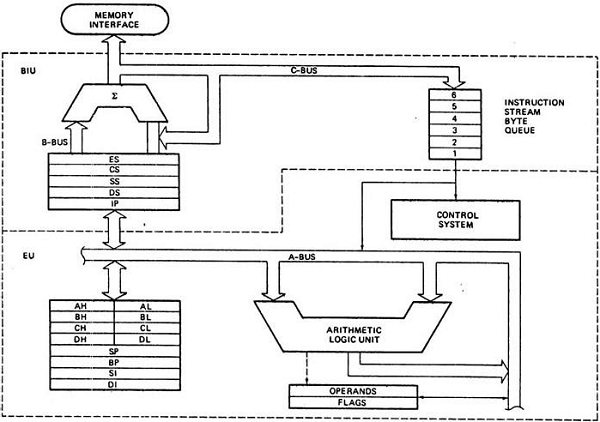
**Aim** : Run the simulator using the given link [https://yjdoc2.github.io/8086- emulator-web/compile](https://yjdoc2.github.io/8086-). Study and execute First 6 examples from the repository <https://github.com/YJDoc2/8086-Emulator/tree/master/examples>

**Submission Sheet**

| SAP ID | Name of Student | Date of Experiment | Date of Submission | Remarks |
| --- | --- | --- | --- | --- |
| 60004190057 | Junaid Girkar |  |  |  |

## **Architecture of 8086**

The following diagram depicts the architecture of a 8086 Microprocessor −



**Pins on 8086:**

**AD0-AD15 (Address Data Bus):** Bidirectional address/data lines. These are low order address bus. They are multiplexed with data.

When these lines are used to transmit memory address, the symbol A is used instead of AD, for example, A0- A15.

**A16 - A19 (Output):** High order address lines. These are multiplexed with status signals.

**A16/S3, A17/S4: A16 and A17** are multiplexed with segment identifier signals S3 and S4.

**A18/S5: A18** is multiplexed with interrupt status S5.

**A19/S6: A19** is multiplexed with status signal S6.

**BHE/S7 (Output**): Bus High Enable/Status. During T1, it is low. It enables the data onto the most significant half of the data bus, D8-D15. 8-bit devices connected to the upper half of the data bus use BHE signal. It is multiplexed with status signal S7. S7 signal is available during T3 and T4.

**RD** (Read): For read operation. It is an output signal. It is active when LOW.

**Ready** (Input): The addressed memory or I/O sends acknowledgment through this pin. When HIGH, it denotes that the peripheral is ready to transfer data.

**RESET** (Input): System reset. The signal is active HIGH.

**CLK** (input): Clock 5, 8 or 10 MHz.

**INTR**: Interrupt Request.

**NMI** (Input): Non-maskable interrupt request.

**TEST** (Input): Wait for test control. When LOW the microprocessor continues execution otherwise waits.

**VCC**: Power supply +5V dc.

**GND**: Ground.

## Operating Modes of 8086

There are two operating modes of operation for Intel 8086, namely the minimum mode and the maximum mode.

When only one 8086 CPU is to be used in a microprocessor system, the 8086 is used in the Minimum mode of operation.

In a multiprocessor system 8086 operates in the Maximum mode.

**ADDRESSING**:

The 8086 microprocessors have 8 addressing modes. Two addressing modes have been provided for instructions which operate on register or immediate data.

**These two addressing modes are:**

**Register Addressing**: In register addressing, the operand is placed in one of the 16-bit or 8-bit general purpose registers.

Example

* MOV AX, CX
* ADD AL, BL
* ADD CX, DX

**Immediate Addressing**: In immediate addressing, the operand is specified in the instruction itself.

Example

* MOV AL, 35H
* MOV BX, 0301H
* MOV [0401], 3598H
* ADD AX, 4836H

The remaining 6 addressing modes specify the location of an operand which is placed in a memory.

These 6 addressing modes are:

**Direct Addressing**: In direct addressing mode, the operand?s offset is given in the instruction as an 8-bit or 16-bit displacement element.

Example

* ADD AL, [0301]

The instruction adds the content of the offset address 0301 to AL. the operand is placed at the given offset (0301) within the data segment DS.

**Register Indirect Addressing**: The operand's offset is placed in any one of the registers BX, BP, SI or DI as specified in the instruction.

Example

* MOV AX, [BX]

It moves the contents of memory locations addressed by the register BX to the register AX.

**Based Addressing**: The operand's offset is the sum of an 8-bit or 16-bit displacement and the contents of the base register BX or BP. BX is used as base register for data segment, and the BP is used as a base register for stack segment.

Effective address (Offset) = [BX + 8-bit or 16-bit displacement].

Example

* MOV AL, [BX+05]; an example of 8-bit displacement.
* MOV AL, [BX + 1346H]; example of 16-bit displacement.

**Indexed Addressing**: The offset of an operand is the sum of the content of an index register SI or DI and an 8-bit or 16-bit displacement.

Offset (Effective Address) = [SI or DI + 8-bit or 16-bit displacement]

Example

* MOV AX, [SI + 05]; 8-bit displacement.
* MOV AX, [SI + 1528H]; 16-bit displacement.

**Based Indexed Addressing**: The offset of operand is the sum of the content of a base register BX or BP and an index register SI or DI.

Effective Address (Offset) = [BX or BP] + [SI or DI]

Here, BX is used for a base register for data segment, and BP is used as a base register for stack segment.

Example

* ADD AX, [BX + SI]
* MOV CX, [BX + SI]

**Based Indexed with Displacement**: In this mode of addressing, the operand's offset is given by:

Effective Address (Offset) = [BX or BP] + [SI or DI] + 8-bit or 16-bit displacement

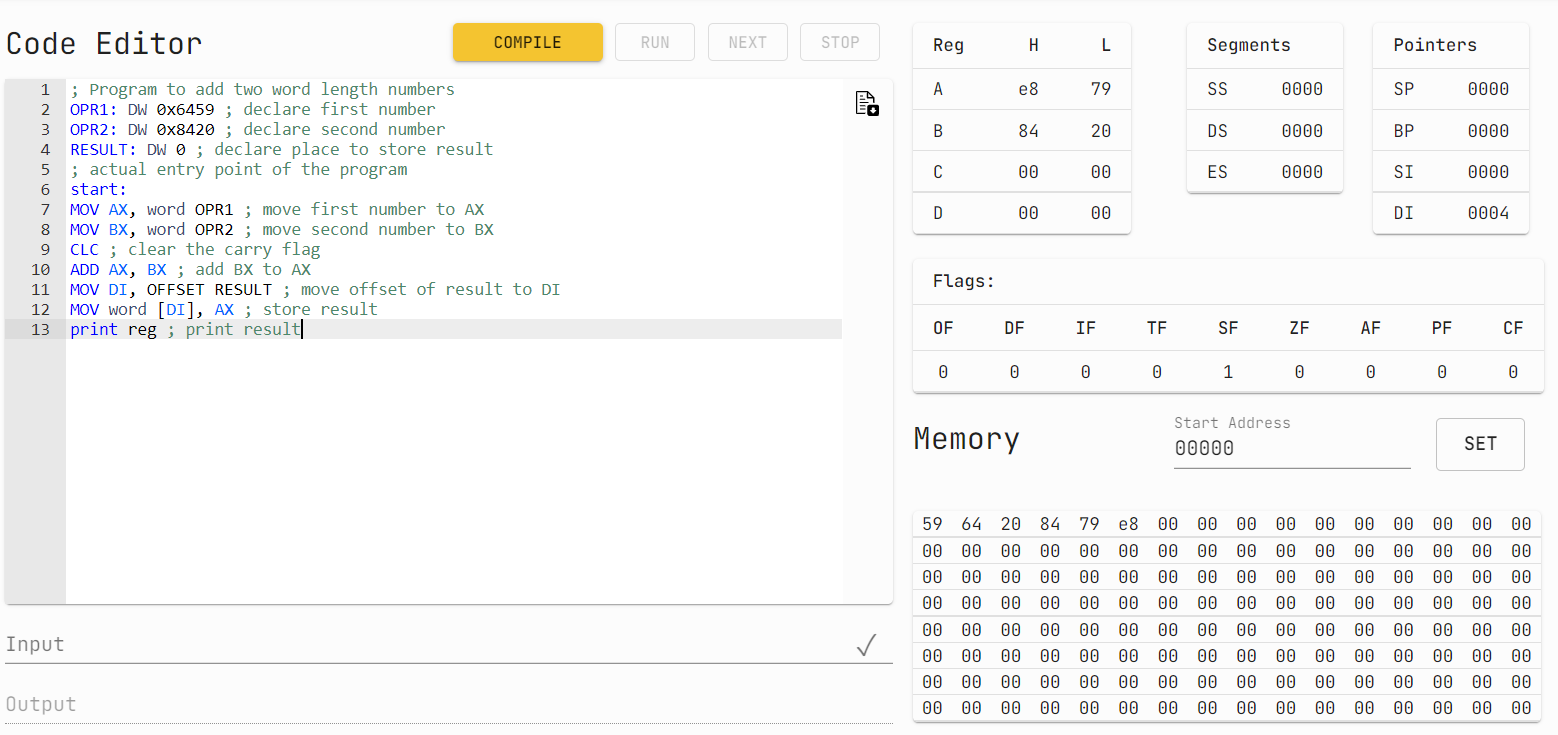
Example

* MOV AX, [BX + SI + 05]; 8-bit displacement
* MOV AX, [BX + SI + 1235H]; 16-bit displacement

**1. Program to add two word length number**

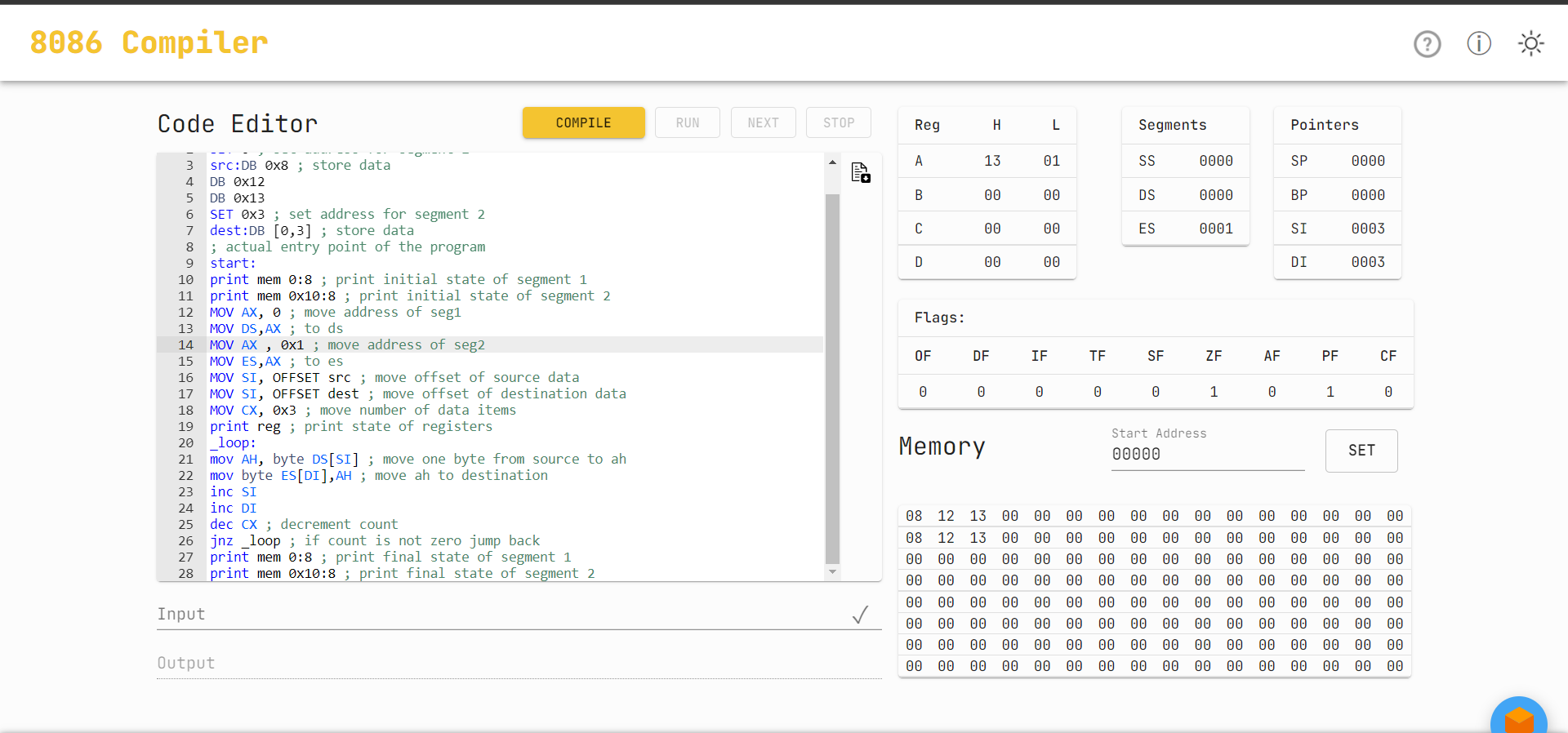


| ; Program to add two word length numbers OPR1: DW 0x6459 ; declare first number OPR2: DW 0x8420 ; declare second number RESULT: DW 0 ; declare place to store result ; actual entry point of the program start: MOV AX, word OPR1 ; move first number to AX MOV BX, word OPR2 ; move second number to BX CLC ; clear the carry flag ADD AX, BX ; add BX to AX MOV DI, OFFSET RESULT ; move offset of result to DI MOV word [DI], AX ; store result print reg ; print result |
| --- |



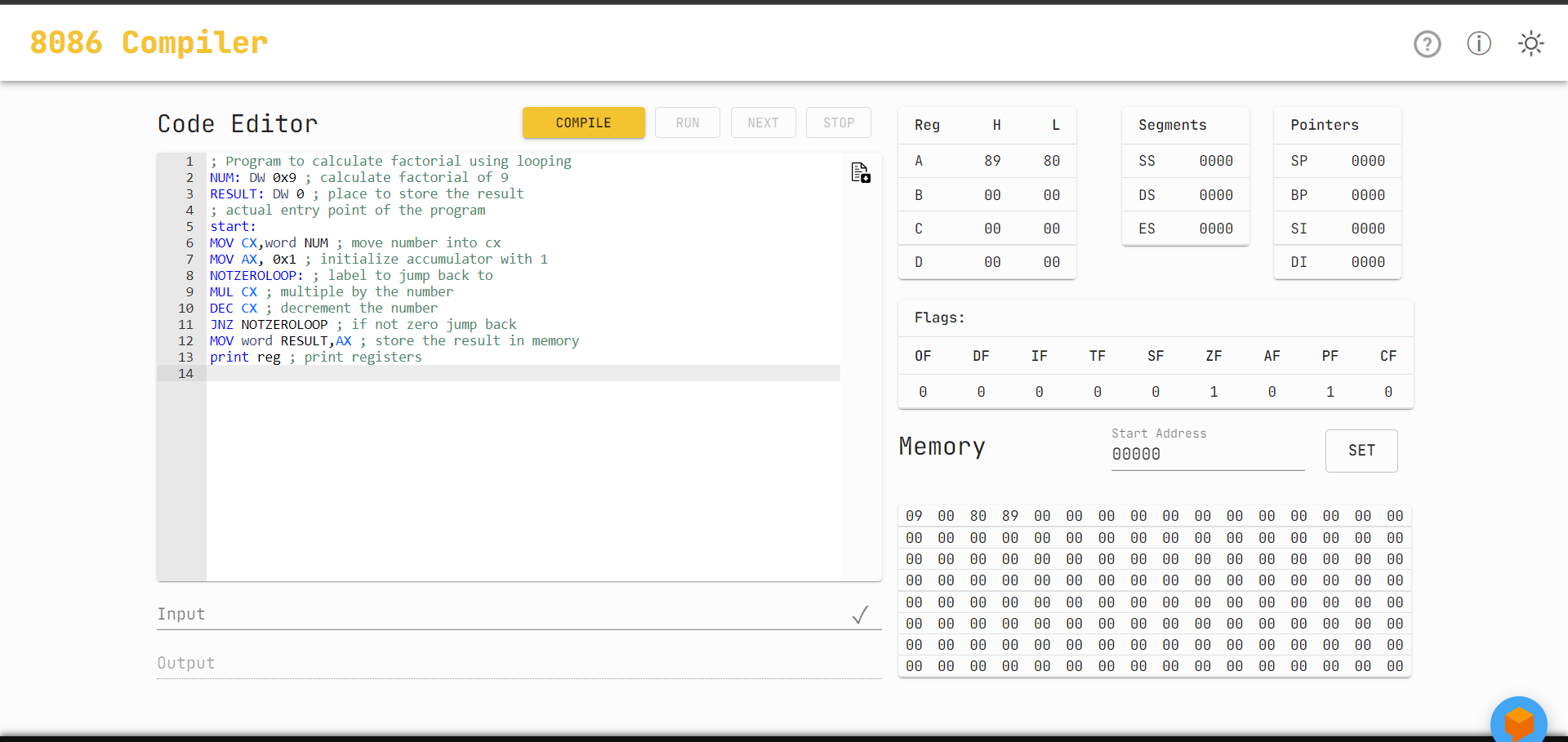
**2. A Program to move data from one segment to another.**

| ; A Program to move data from one segment to another SET 0 ; set address for segment 1 src:DB 0x8 ; store data DB 0x12 DB 0x13 SET 0x3 ; set address for segment 2 dest:DB [0,3] ; store data ; actual entry point of the program start: print mem 0:8 ; print initial state of segment 1 print mem 0x10:8 ; print initial state of segment 2 MOV AX, 0 ; move address of seg1 MOV DS,AX ; to ds MOV AX , 0x1 ; move address of seg2 MOV ES,AX ; to es MOV SI, OFFSET src ; move offset of source data MOV SI, OFFSET dest ; move offset of destination data MOV CX, 0x3 ; move number of data items print reg ; print state of registers \_loop: mov AH, byte DS[SI] ; move one byte from source to ah mov byte ES[DI],AH ; move ah to destination inc SI inc DI dec CX ; decrement count jnz \_loop ; if count is not zero jump back print mem 0:8 ; print final state of segment 1 print mem 0x10:8 ; print final state of segment 2 |
| --- |



**3. Program to calculate factorial using looping.**

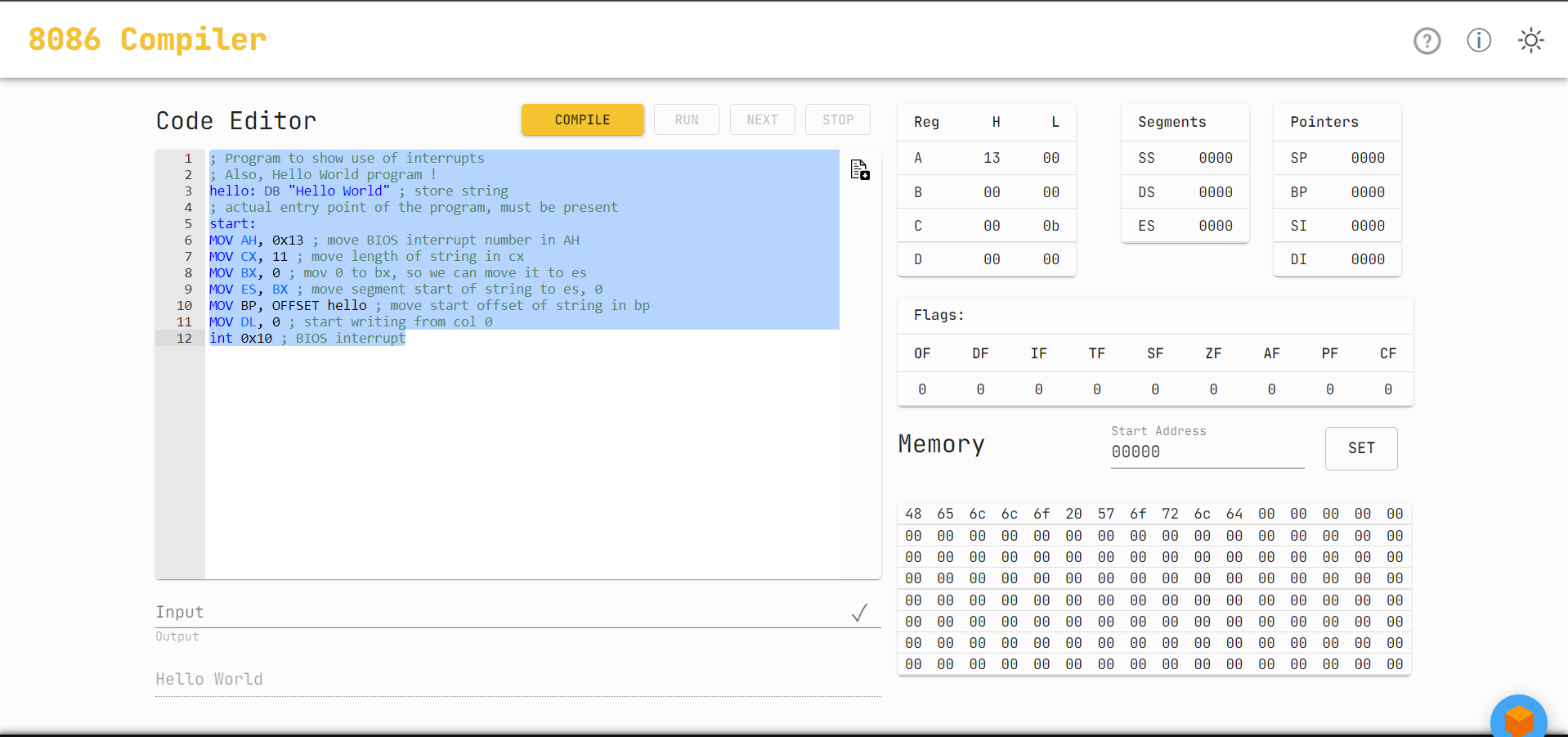
| ; Program to calculate factorial using looping NUM: DW 0x9 ; calculate factorial of 9 RESULT: DW 0 ; place to store the result ; actual entry point of the program start: MOV CX,word NUM ; move number into cx MOV AX, 0x1 ; initialize accumulator with 1 NOTZEROLOOP: ; label to jump back to MUL CX ; multiple by the number DEC CX ; decrement the number JNZ NOTZEROLOOP ; if not zero jump back MOV word RESULT,AX ; store the result in memory print reg ; print registers |
| --- |



**4. Program to show use of interrupts.**

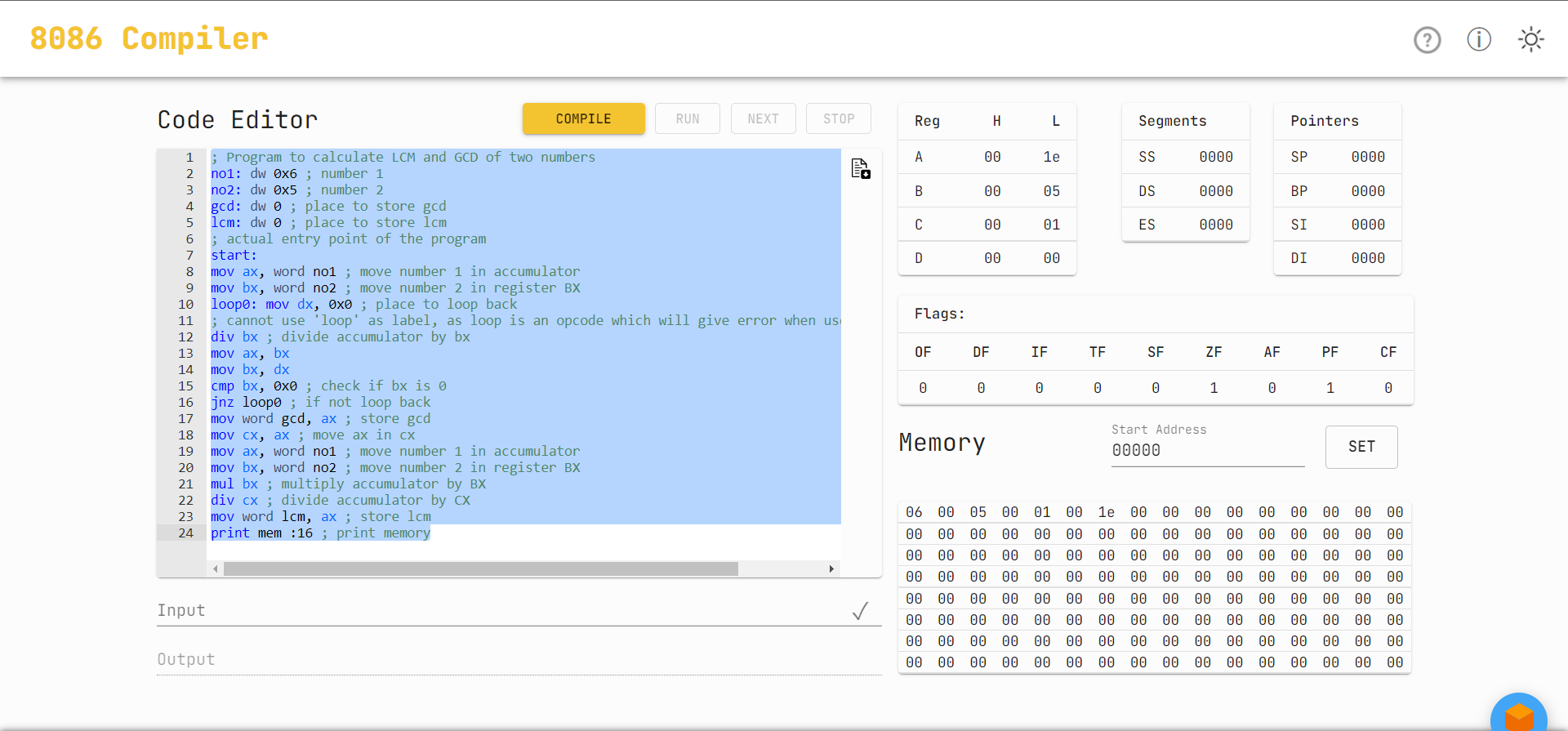


| ; Program to show use of interrupts ; Also, Hello World program ! hello: DB "Hello World" ; store string ; actual entry point of the program, must be present start: MOV AH, 0x13 ; move BIOS interrupt number in AH MOV CX, 11 ; move length of string in cx MOV BX, 0 ; mov 0 to bx, so we can move it to es MOV ES, BX ; move segment start of string to es, 0 MOV BP, OFFSET hello ; move start offset of string in bp MOV DL, 0 ; start writing from col 0 int 0x10 ; BIOS interrupt |
| --- |



**5. Program to calculate LCM and GCD of two numbers.**

| ; Program to calculate LCM and GCD of two numbers no1: dw 0x6 ; number 1 no2: dw 0x5 ; number 2 gcd: dw 0 ; place to store gcd lcm: dw 0 ; place to store lcm ; actual entry point of the program start: mov ax, word no1 ; move number 1 in accumulator mov bx, word no2 ; move number 2 in register BX loop0: mov dx, 0x0 ; place to loop back ; cannot use 'loop' as label, as loop is an opcode which will give error when used with jumps div bx ; divide accumulator by bx mov ax, bx mov bx, dx cmp bx, 0x0 ; check if bx is 0 jnz loop0 ; if not loop back mov word gcd, ax ; store gcd mov cx, ax ; move ax in cx mov ax, word no1 ; move number 1 in accumulator mov bx, word no2 ; move number 2 in register BX mul bx ; multiply accumulator by BX div cx ; divide accumulator by CX mov word lcm, ax ; store lcm print mem :16 ; print memory |
| --- |



**6. A program to sort the numbers in descending order, using bubble sort**

| ; A program to sort the numbers in descending order, using bubble sort vals:DB 0xF ; declaration of the numbers DB 0x5A DB 0x24 DB 0x2 DB 0x56 last: DB 0 ; declaring an element to get total number of elements later ; actual entry point of the program start: print mem :8 ; print initial state of memory MOV CH, OFFSET last ; move number of elements to CH outer: ; loop label for outer loop MOV CL, OFFSET last ; move number of elements to CL MOV SI, OFFSET vals ; move offset of values to si inner: ; loop label for inner loop MOV AX, word [SI] ; move two adjacent numbers to AX CMP AL,AH ; compare both values JNC skip ; jump to skip if first num is greater than second XCHG AL, AH ; exchange both numbers MOV word [SI], AX ; move exchanged numbers to memory skip: INC SI ; increment SI DEC CL ; decrement inner loop counter JNZ inner ; jump back to inner if counter is not zero DEC CH ; decrement outer loop counter JNZ outer ; jump back to outer, if counter is not zero print mem :8 ; print final state of memory |
| --- |

