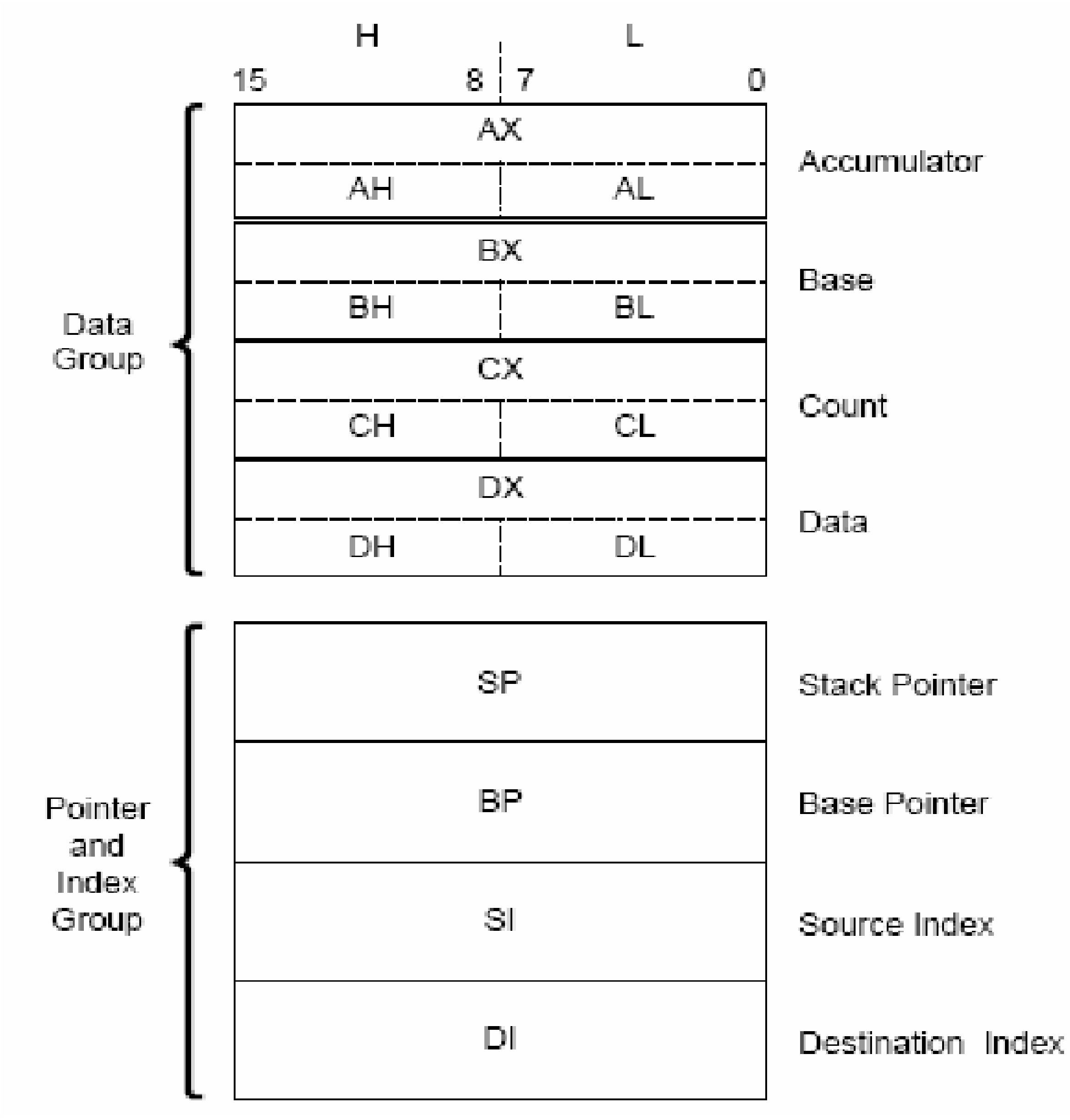
CPU. Register Set

Registers set of I8086

1. *General Purpose Registers*



The data registers can be addressed by their upper or lower halves.

Each data register can be used interchangeably as a 16-bit register or two 8-bit registers.

The pointer and index registers are always accessed as 16-bit values.

The µ**p** can use data registers without constraint in most arithmetic and logic operations. Arithmetic and logic operations can also use the pointer and index registers. Some instructions use certain registers implicitly allowing compact encoding.

SP - Stack Pointer : Always points to top item of the stack.
BP - **Base Pointer**: It is used to access any item in the stack;
SI - Source Index: Contains the address of the current element in the source string;
DI - Destination Index: Contains the address of the current element in the destination string;

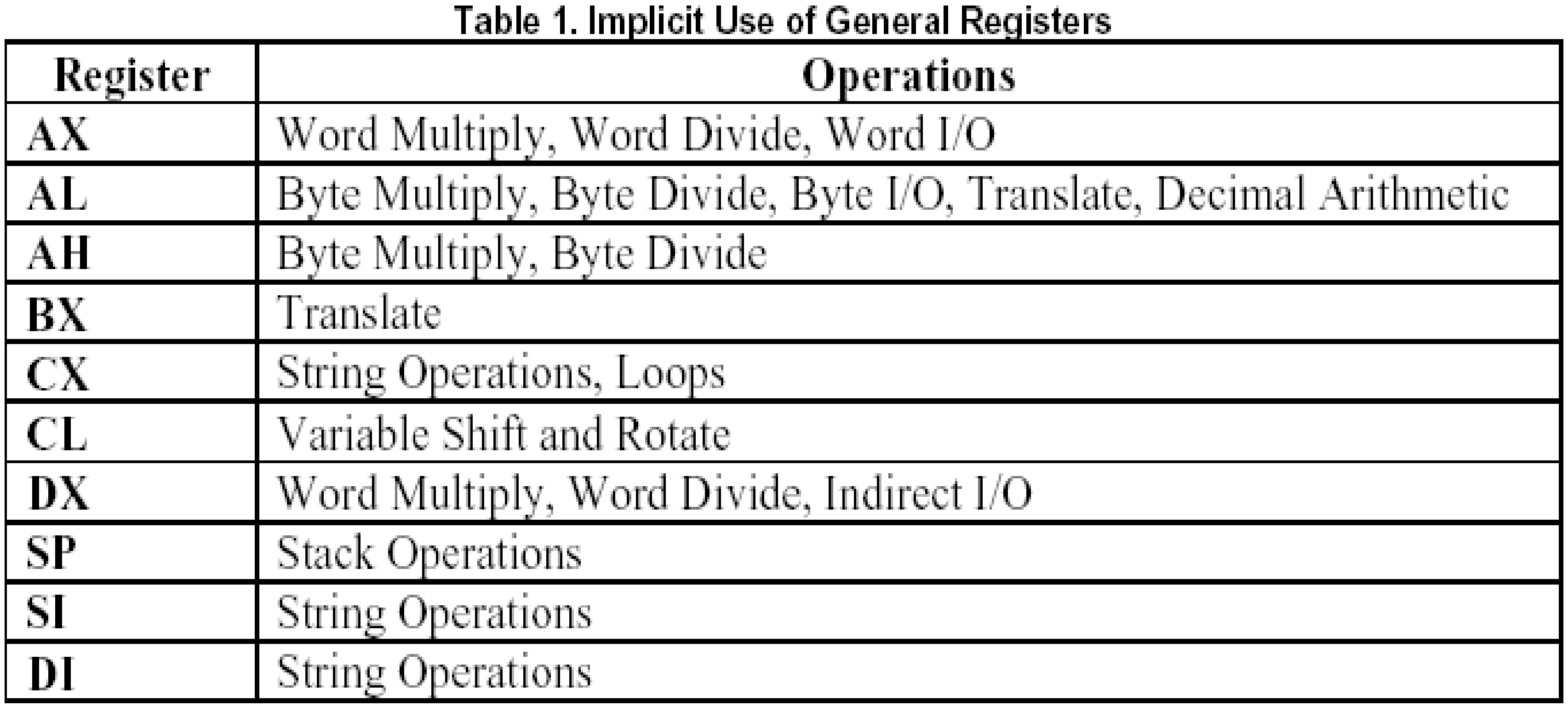


Table 1. Implicit Use of General Registers

Register	Operations
AX	Word Multiply, Word Divide, Word I/O
AL	Byte Multiply, Byte Divide, Byte I/O, Translate, Decimal Arithmetic
AH	Byte Multiply, Byte Divide
BX	Translate
CX	String Operations, Loops
CL	Variable Shift and Rotate
DX	Word Multiply, Word Divide, Indirect I/O
SP	Stack Operations
SI	String Operations
DI	String Operations

2. *Segment registers*

The mp 8086 has a 20-bit address bus for 1 Mbyte external memory but inside the CPU registers have 16 bits that can access 64 Kbytes.

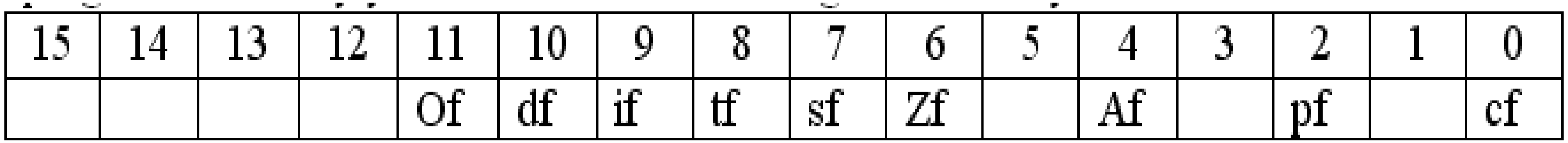
The 8086 family memory space is divided into logical segments of up to 64 Kbytes each. The segment registers contain the base addresses (starting locations) of these memory segments.

- **CS** (code segment) - points at the segment containing the current program.
- **DS** (data segment)- generally points at the segment where variables are defined.
- **ES** (extra segment)- extra segment register, it's up to a coder to define its usage.
- **SS** (stack segment)- points at the segment containing the stack.

3. Special purpose registers

IP - the instruction pointer or program counter: Always points to next instruction to be executed. It contains the offset (displacement) of the next instruction from the start address of the code segment.

Flags Register - determines the current state of the processor. From 16 bits are used only 9.



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
				Of	df	if	tf	sf	Zf		Af		pf		cf

Condition flags:

0 bit -Carry Flag (CF) - this flag is set to **1** when there is a carry (borrow) from the 8 or 16 bit in addition or subtraction operation.

2 bit - Parity Flag (PF) - this flag is set to **1** when there is even number of one bits in result, and to **0** when there is odd number of one bits. Even if result is a word only 8 low bits are analyzed!

4 bit - Auxiliary Flag (AF) - set to **1** when there is an **unsigned overflow** for low nibble (4 bits).

6 bit - Zero Flag (ZF) - set to **1** when result is **zero**. For none zero result this flag is set to **0**.

7 bit - Sign Flag (SF) - set to **1** when result is **negative**. When result is **positive** it is set to **0**. Actually this flag take the value of the most significant bit.

11 bit - Overflow Flag (OF) - set to **1** when there is a **signed overflow**.

Control flags:

8 bit - Trap Flag (TF) System flag - Used for on-chip debugging when TF=1. In this case the interrupt is generated (int 1) which calls a special routine to show the state of internal registers.

9 bit - Interrupt enable Flag (IF) System flag - when this flag is set to **1** CPU reacts to interrupts on INTR input of the mp from external devices. When IF=0 interrupts are not allowed (masked). IF do not react to NMI (non maskable) interrupts and to internal interrupts performed by instruction INT.

Instructions CLI (clear interrupt) and STI (set interrupt) are used to control this flag.

10 bit - Direction Flag (DF) - this flag is used by some instructions to process data chains, when this flag is set to **0** - the processing is done forward (increment of SI and DI registers), when this flag is set to **1** the processing is done backward - decrement

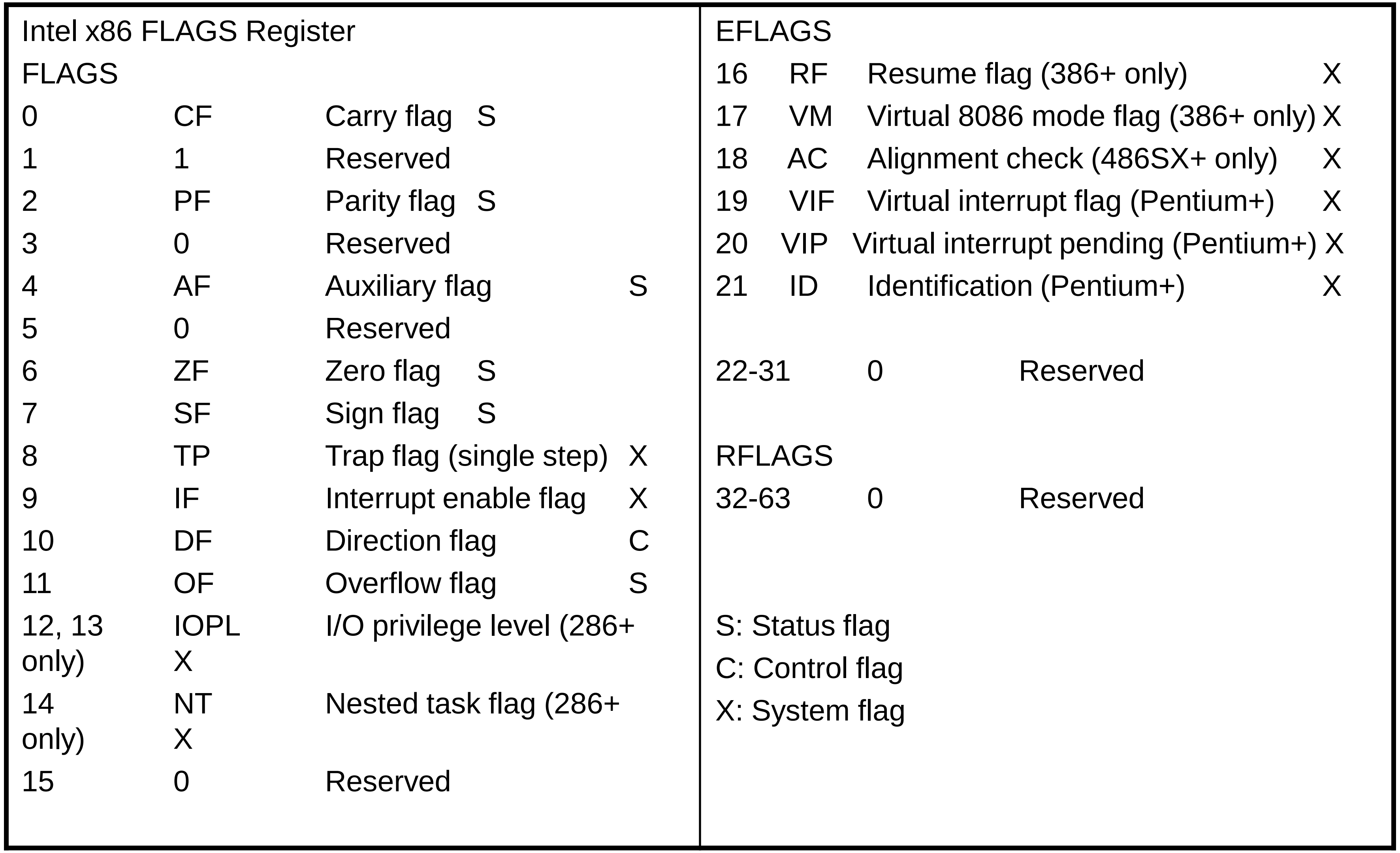
(instructions CLD and STD).

Exercises

Determine the value of CF, ZF, SF, OF, PF and AF after the following addition operations:

1. 342Ah+57E2h=8C0Ch
2. E42Ah+96B8h=7AE2h
3. C739h+38C7h=0000h
4. F502h+1A7h =F6A9h
5. 6BD3h+90F1h=FCC4h

The FLAGS register is the status register in Intel x86 microprocessors that contains the current state of the processor. This register is 16 bits wide. Its successors, the EFLAGS and RFLAGS registers are 32 bits and 64 bits wide, respectively. The wider registers retain compatibility with their smaller predecessors.



Intel x86 FLAGS Register

FLAGS

0	CF	Carry flag	S
1	1	Reserved	
2	PF	Parity flag	S
3	0	Reserved	
4	AF	Auxiliary flag	S
5	0	Reserved	
6	ZF	Zero flag	S
7	SF	Sign flag	S
8	TP	Trap flag (single step)	X
9	IF	Interrupt enable flag	X
10	DF	Direction flag	C
11	OF	Overflow flag	S
12, 13	IOPL	I/O privilege level (286+ only)	X
14	NT	Nested task flag (286+ only)	X
15	0	Reserved	

EFLAGS

16	RF	Resume flag (386+ only)	X
17	VM	Virtual 8086 mode flag (386+ only)	X
18	AC	Alignment check (486SX+ only)	X
19	VIF	Virtual interrupt flag (Pentium+)	X
20	VIP	Virtual interrupt pending (Pentium+)	X
21	ID	Identification (Pentium+)	X
22-31	0	Reserved	

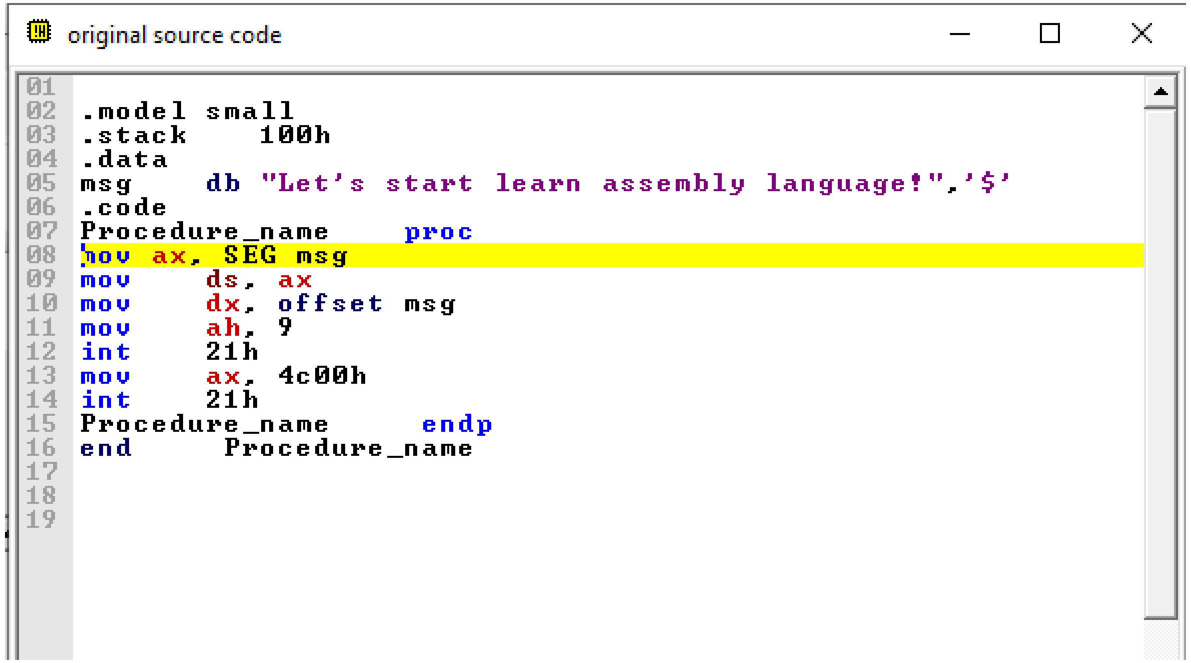
RFLAGS

32-63	0	Reserved

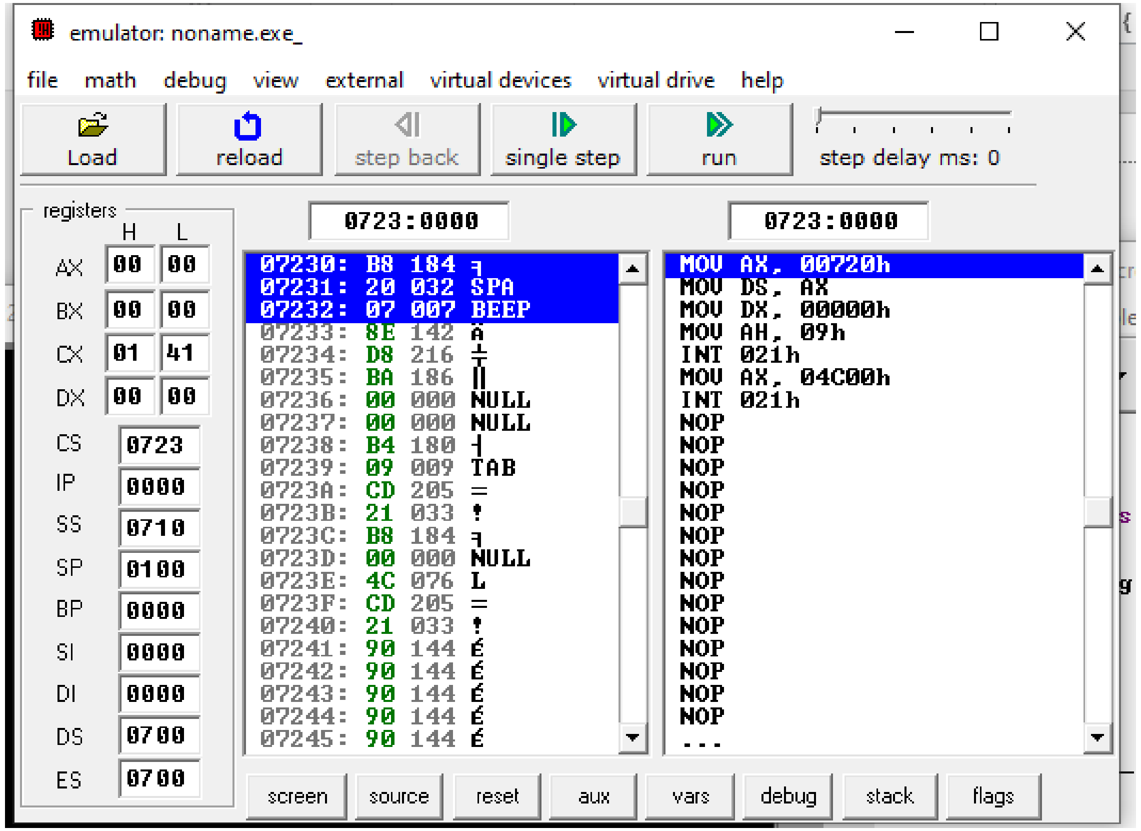
S: Status flag
C: Control flag
X: System flag

Code exapmles

```
.model small
.stack        100h
.data
msg           db "Let's start learn assembly language!",'$‘
.code
Procedure_name            proc
mov ax, SEG msg
mov           ds, ax
mov           dx, offset msg
mov           ah, 9
int           21h
mov           ax, 4c00h
int           21h
Procedure_name            endp
end            Procedure_name
```



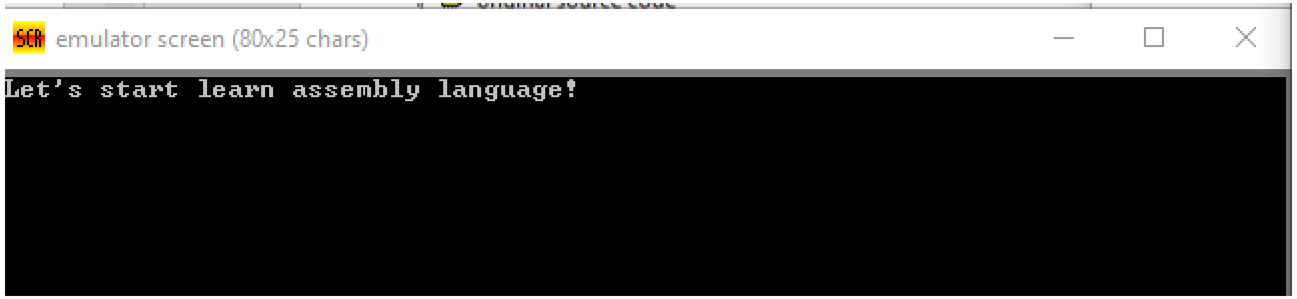
```
.model small
.stack    100h
.data
msg    db "Let’s start learn assembly language!",’$’
.code
Procedure_name     proc
mov ax, SEG msg
mov      ds, ax
mov      dx, offset msg
mov      ah, 9
int      21h
mov      ax, 4c00h
int      21h
Procedure_name      endp
end       Procedure_name
```



	H	L
AX	00	00
BX	00	00
CX	01	41
DX	00	00
CS	0723	
IP	0000	
SS	0710	
SP	0100	
BP	0000	
SI	0000	
DI	0000	
DS	0700	
ES	0700	

```
0723:0000
07230: B8 184 ╕
07231: 20 032 SPA
07232: 07 007 BEEP
07233: 8E 142 Ä
07234: D8 216 ╪
07235: BA 186 ║
07236: 00 000 NULL
07237: 00 000 NULL
07238: B4 180 ┤
07239: 09 009 TAB
0723A: CD 205 ═
0723B: 21 033 !
0723C: B8 184 ╕
0723D: 00 000 NULL
0723E: 4C 076 L
0723F: CD 205 ═
07240: 21 033 !
07241: 90 144 É
07242: 90 144 É
07243: 90 144 É
07244: 90 144 É
07245: 90 144 É
```

```
0723:0000
MOV AX, 00720h
MOV DS, AX
MOV DX, 00000h
MOV AH, 09h
INT 021h
MOV AX, 04C00h
INT 021h
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
NOP
...
```



```
Let’s start learn assembly language!
```