Lab 1

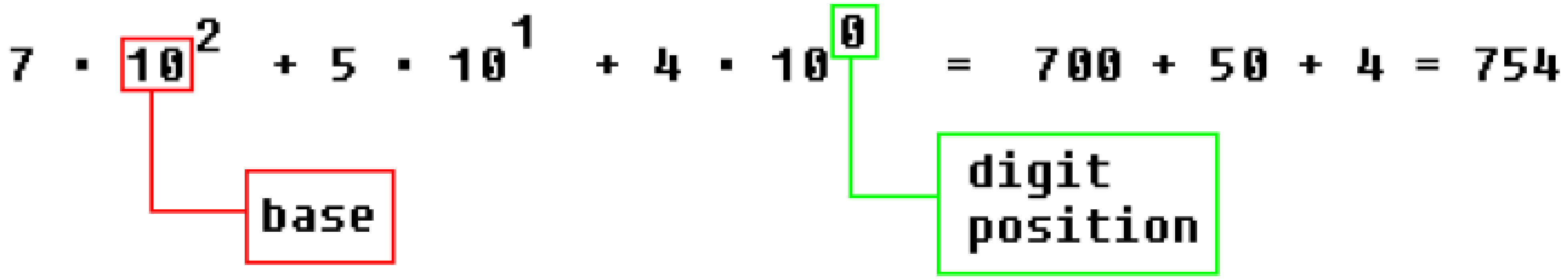
Numbering System
8086 architecture
Emulator 8086

Decimal System

In the decimal system there are **10 digits**:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

EX: 754


$$7 \cdot \boxed{10}^2 + 5 \cdot 10^1 + 4 \cdot 10^{\boxed{0}} = 700 + 50 + 4 = 754$$

Decimal System

Important note: any number in power of zero is 1, even zero in power of zero is 1:

$$\boxed{10^0 = 1} \quad \boxed{0^0 = 1} \quad \boxed{x^0 = 1}$$

Binary System

Computers are not as smart as humans, it's easy to make an electronic machine with two states: **on** and **off**, or **1** and **0**.

Computers use binary system, binary system uses 2 digits:

0, 1

And thus the **base** is **2**.

Binary System

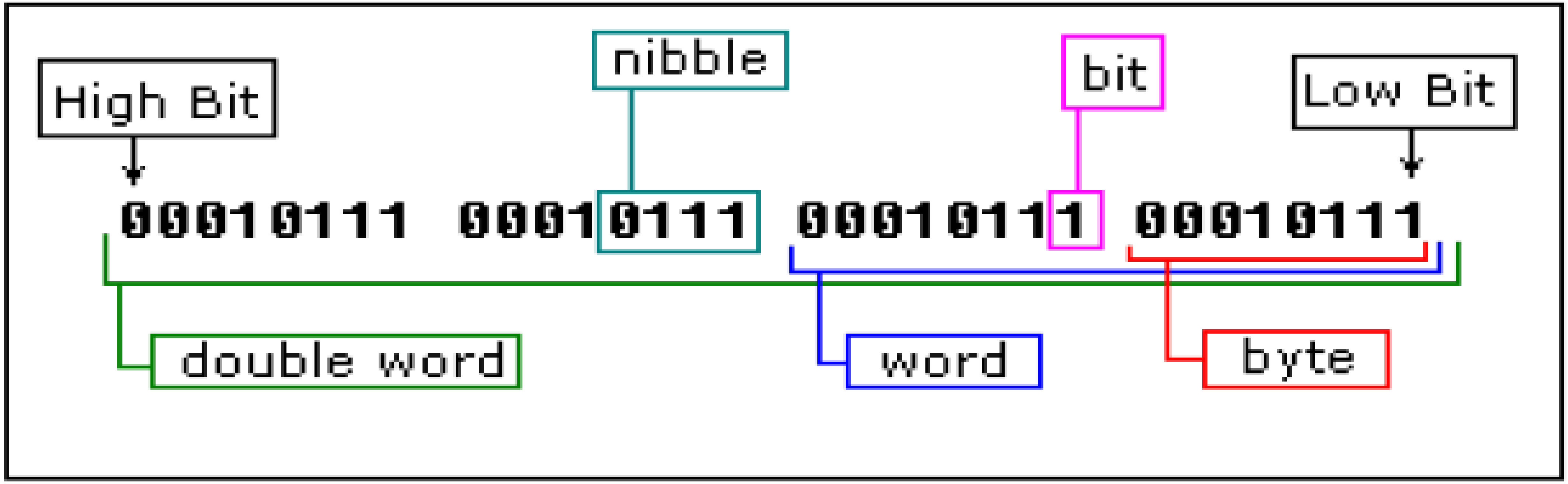
Each **digit** in a binary number is called a **BIT**,

4 bits form a **NIBBLE**,

8 bits form a **BYTE**,

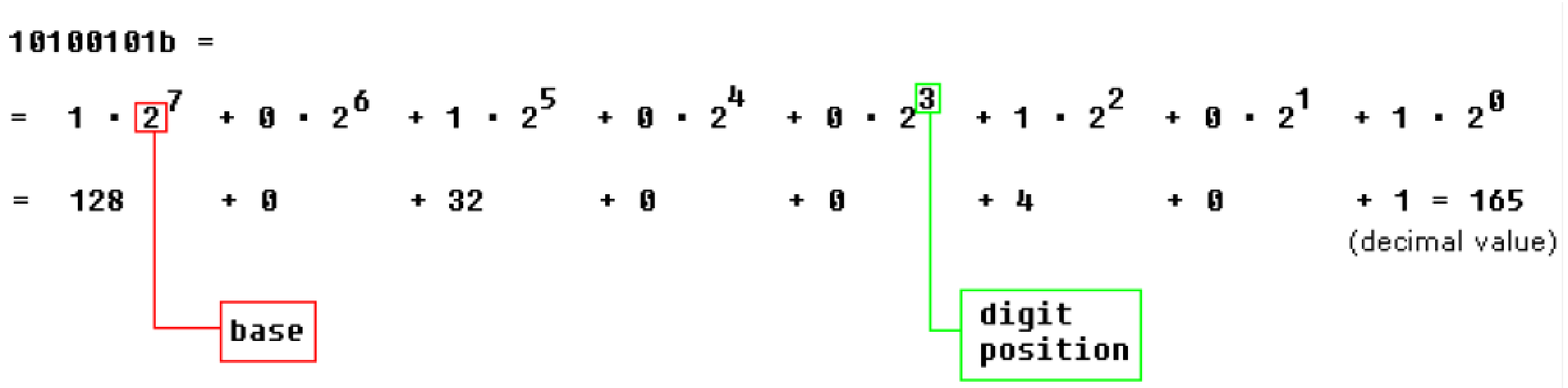
two bytes form a **WORD**,

two words form a **DOUBLE WORD.**


High Bit
nibble
bit
Low Bit
00010111 00010111 00010111 00010111
double word
word
byte

Binary System

There is a convention to add **"b"** in the end of a binary number, this way we can determine that 101b is a binary number with decimal value of 5.



$$
\begin{aligned}
10100101b &= \\
&= 1 \cdot \boxed{2}^7 + 0 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^{\boxed{3}} + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 \\
&= 128 + 0 + 32 + 0 + 0 + 4 + 0 + 1 = 165
\end{aligned}
$$

(decimal value)

ADD in Binary System

0+0=0

1+0=1

0+1=1

1+1= 0 carry 1

EX: 1011b +1010b

1011b+1111b

SUB in Binary System

0-0=0

1-0=1

0-1=1 borrow 1

1-1= 0

EX: 1011b -1010b=0001b

1100b-1011b=0001b

Hexadecimal System

Hexadecimal System uses **16 digits**:

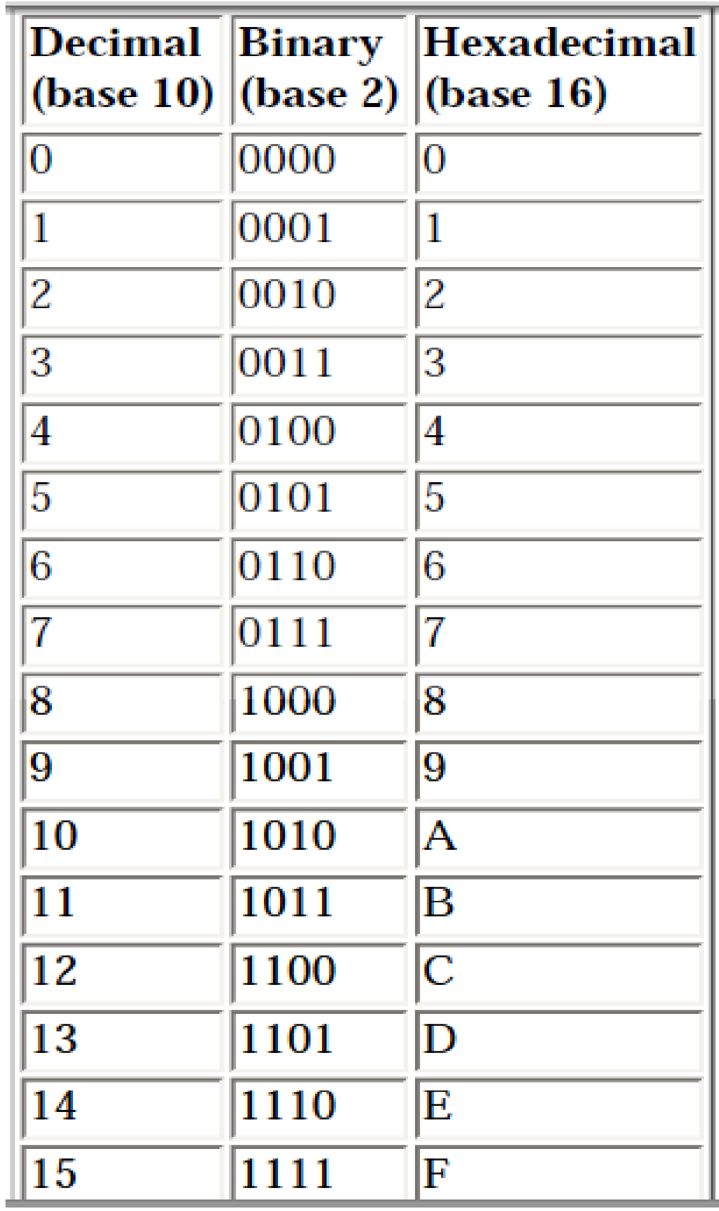
0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

And thus the **base** is **16**.

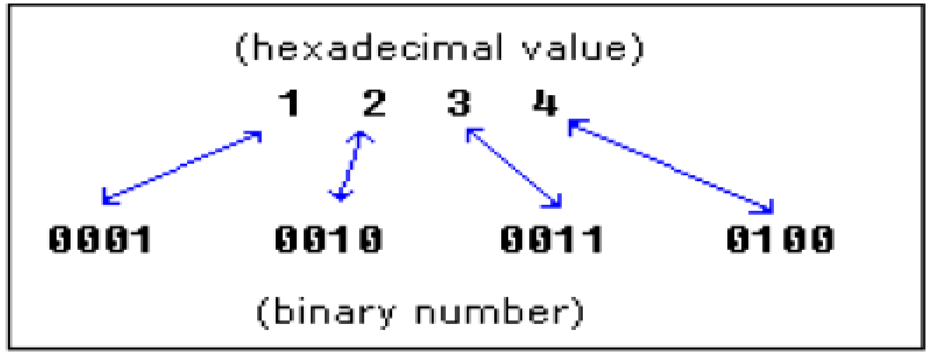
Note:

- Hexadecimal numbers are compact and easy to read.
- It is very easy to convert numbers from binary system to hexadecimal system and vice-versa, every nibble (4 bits)

converted to a hexadecimal digit using this table:



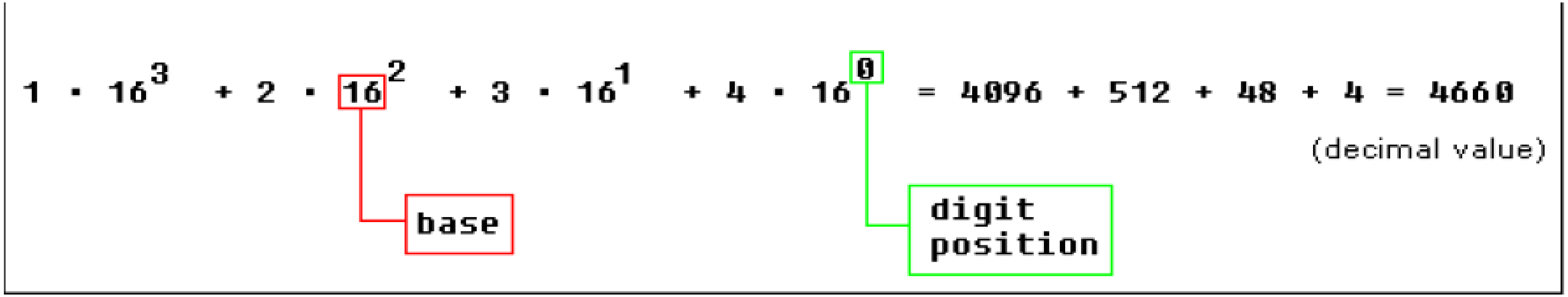
Decimal (base 10)	Binary (base 2)	Hexadecimal (base 16)
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F



Hexadecimal System

- There is a convention to add **"h"** in the end of a hexadecimal number, We also add **"0"** (zero) in the beginning of hexadecimal numbers that begin with a letter (A..F), for example **0E120h**.

- The hexadecimal number **1234h** is equal to decimal value of 4660:



$$1 \cdot 16^3 + 2 \cdot \boxed{16}^2 + 3 \cdot 16^1 + 4 \cdot 16^{\boxed{0}} = 4096 + 512 + 48 + 4 = 4660$$

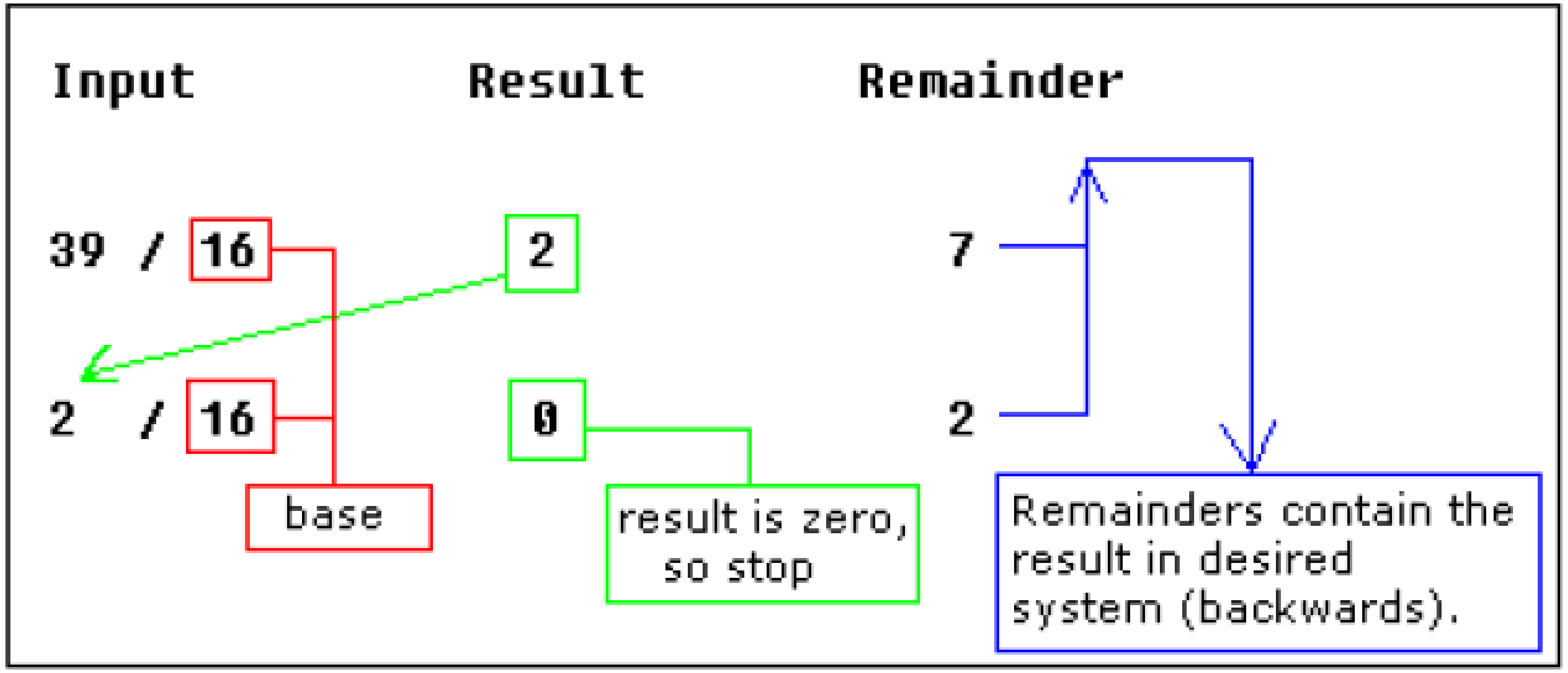
Converting from Decimal System to Other System

In order to convert from decimal system, to any other system, it is required to divide the decimal value by the **base** of the desired system, each time you should remember the **result** and keep the **remainder**, the divide process continues until the **result** is zero.

The **remainders** are then used to represent a value in that system.

Let's convert the value of **39** (base 10) to *Hexadecimal System* (base 16):

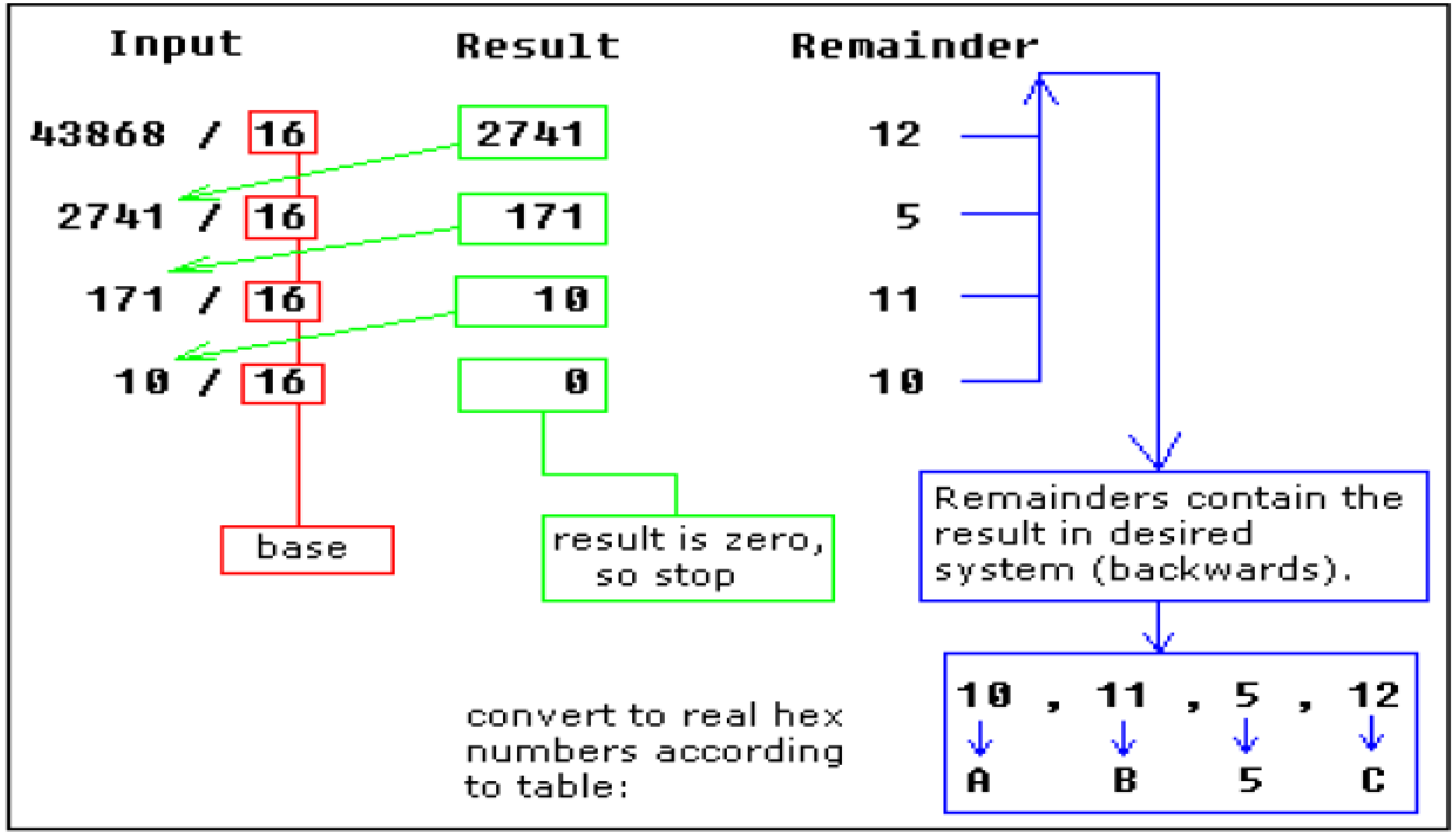
Converting from Decimal System to hexa.



As you see we got this hexadecimal number: **27h**.

Converting from Decimal System to Any Other

let's convert decimal number **43868** to hexadecimal form:



Converting from Decimal System to Any Other

$420.625_{10} =$

$420.625_{10} = 420_{10} + .625_{10}$

Division	Quotient	Remainder
420 ÷ 16	26	4
26 ÷ 16	1	10 (or A)
1 ÷ 16	0	1

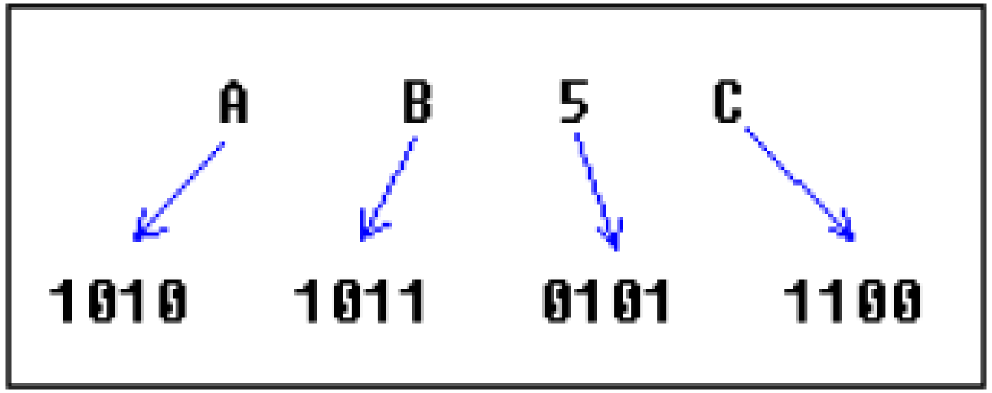
Multiplication	Product	Carry-out
.625 x 16	10.000	10 (or A)

$420.625_{10} = 1A4.A_{16}$

$4135_{10} = 1027_{16}$

$625.625_{10} = 271.A_{16}$

Convert Hexa. to Binary number



As you see we got this binary number: **1010101101011100b**

Number Systems

Binary-Coded Hexadecimal (BCH):

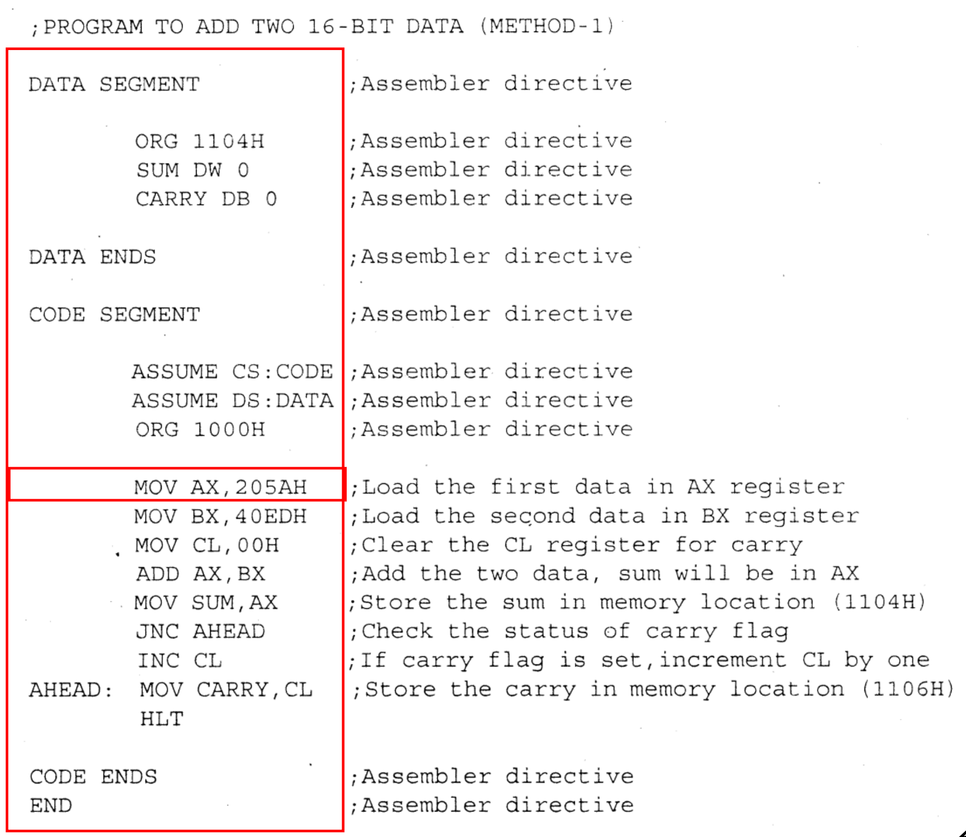
2AC = 0010 1010 1100

1000   0011 1101 . 1110 = 83D.E

Bit A bit is a value of either a 1 or 0 (on or off)
Nibble A Nibble is 4 bits.
Byte a Byte is 8 bits.
A Kilobyte is 1,024 bytes (2^{10}) bytes .
Megabyte (MB) is 1,048,576 bytes or (2^{20}) bytes 1,024 Kilobytes
Gigabyte (GB) is 1,073,741,824 (2^{30}) bytes. 1,024 Megabytes
Terabyte (TB) is (2^{40}) bytes, 1,024 Gigabytes
Petabyte (PB) is (2^{50}) bytes, 1,024 Terabytes
Exabyte (EB) is (2^{60}) bytes, 1,024 Petabytes
Zettabyte (ZB) is (2^{70}) bytes, 1,024 Exabytes
Yottabyte (YB) is (2^{80}) bytes, 1,024 Zettabytes

8086 Microprocessor

Introduction To Assembly Language



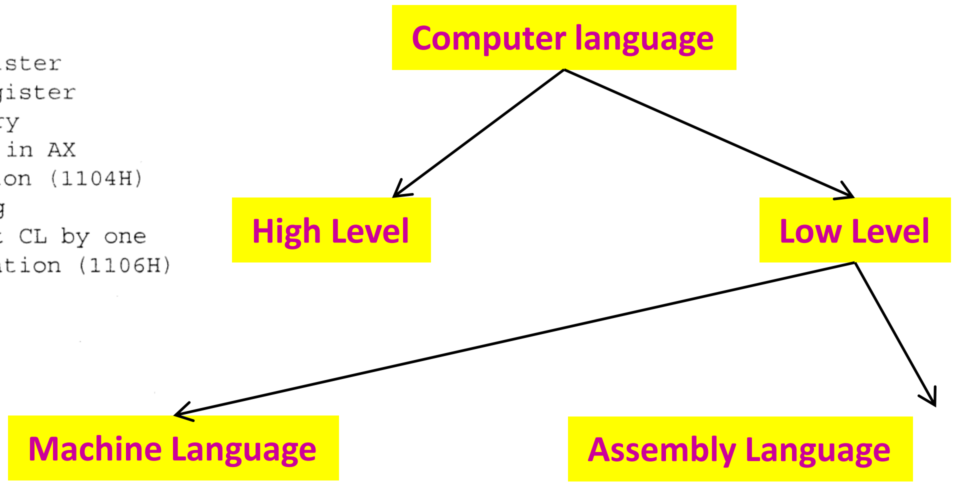
```
;PROGRAM TO ADD TWO 16-BIT DATA (METHOD-1)

DATA SEGMENT              ;Assembler directive

        ORG 1104H         ;Assembler directive
        SUM DW 0          ;Assembler directive
        CARRY DB 0        ;Assembler directive

DATA ENDS                 ;Assembler directive

CODE SEGMENT              ;Assembler directive

        ASSUME CS:CODE    ;Assembler directive
        ASSUME DS:DATA    ;Assembler directive
        ORG 1000H         ;Assembler directive

        MOV AX,205AH      ;Load the first data in AX register
        MOV BX,40EDH      ;Load the second data in BX register
        MOV CL,00H        ;Clear the CL register for carry
        ADD AX,BX         ;Add the two data, sum will be in AX
        MOV SUM,AX        ;Store the sum in memory location (1104H)
        JNC AHEAD         ;Check the status of carry flag
        INC CL            ;If carry flag is set,increment CL by one
AHEAD:   MOV CARRY,CL      ;Store the carry in memory location (1106H)
         HLT

CODE ENDS                 ;Assembler directive
END                       ;Assembler directive
```

Program

A set of instructions written to solve a problem.

Instruction

Directions which a microprocessor follows to execute a task or part of a task.



- Binary bits

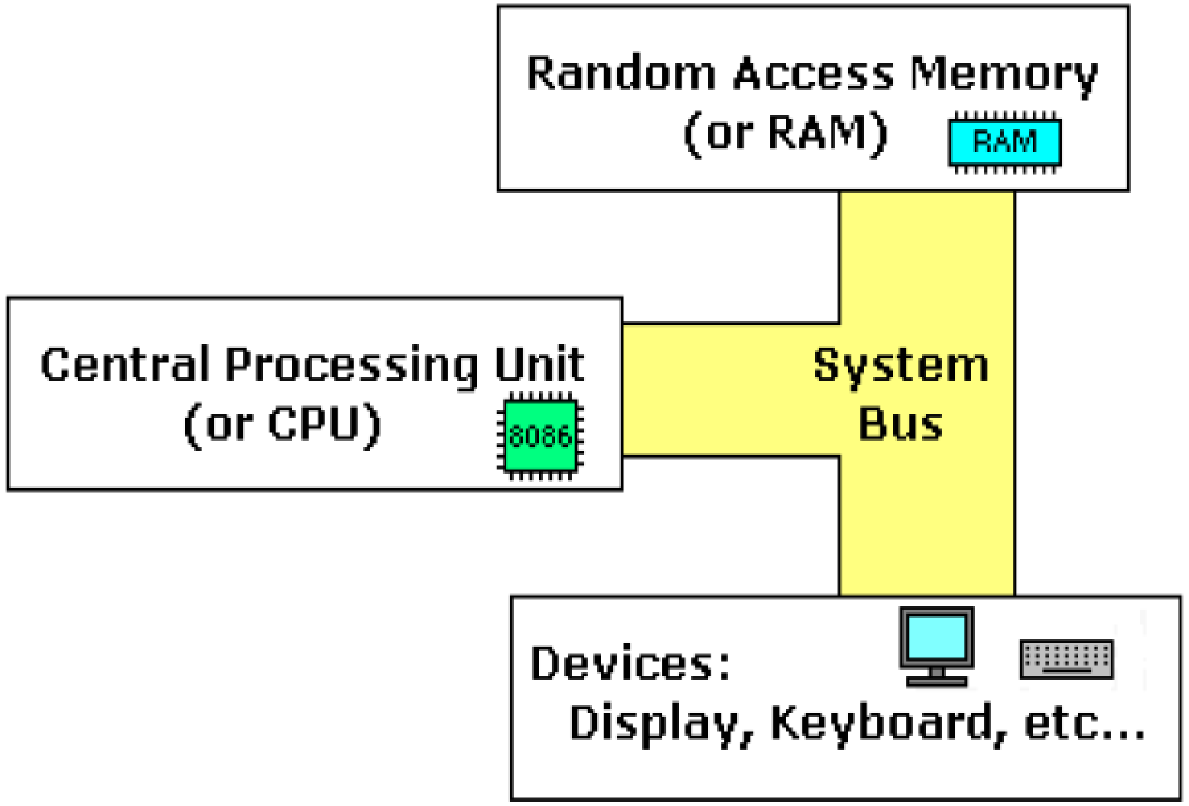
- English Alphabets
- 'Mnemonics'
- Assembler Mnemonics $\rightarrow$ Machine Language

Assembly language is a low-level programming language for a computer, it is converted into executable machine code by a assembler program

Advantages of Assembly Language

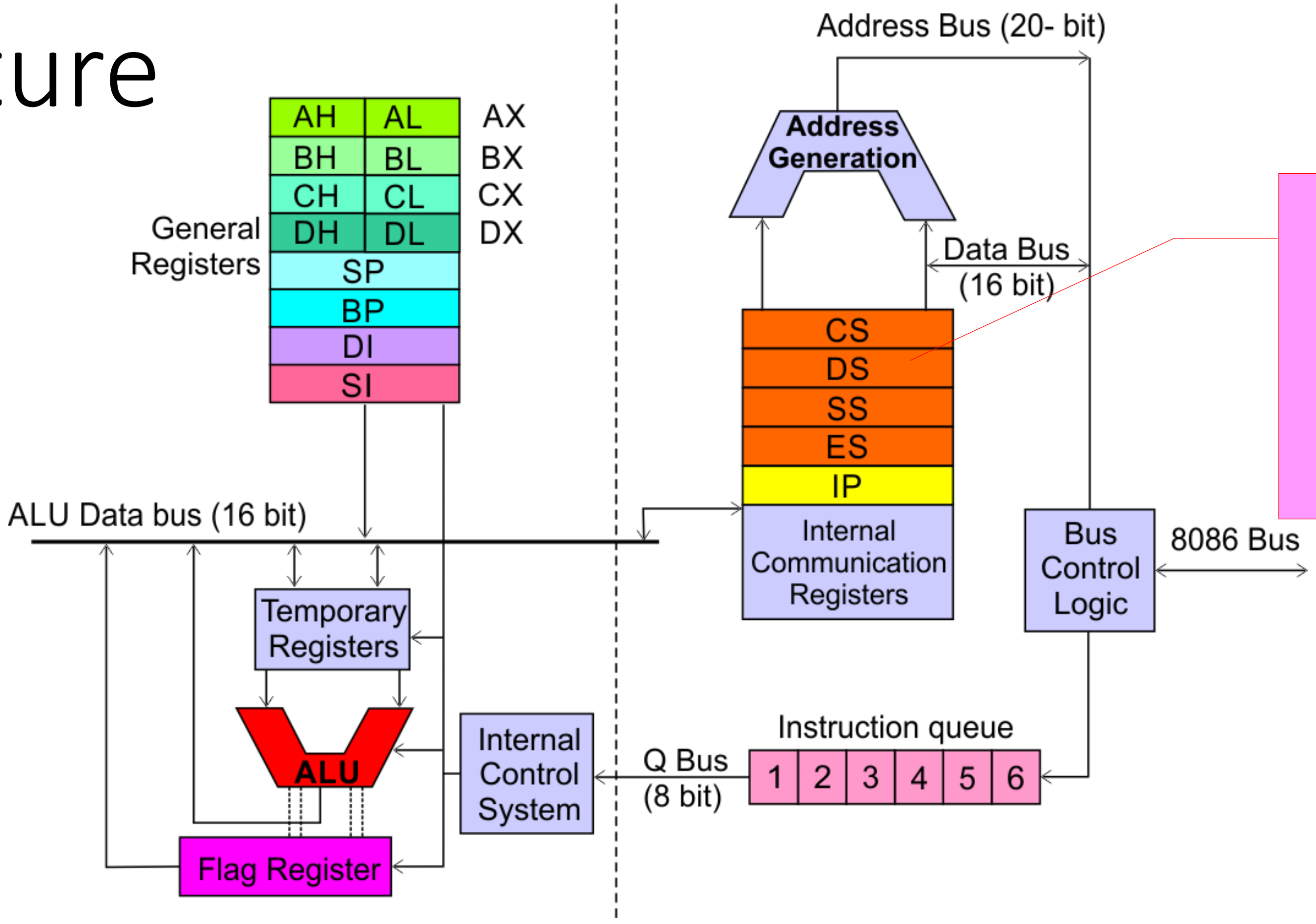
- Having an understanding how programs interface with OS, processor.
- How data is represented in memory.
- How the processor accesses and executes instruction.
- How instructions access and process data.
- It requires less memory and execution time.

Simple Computer Model



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

Architecture



Four 16-bit segment registers

Code Segment (CS)
Data Segment (DS)
Stack Segment (SS)
Extra Segment (ES)

Execution Unit (EU)

EU executes instructions that have already been fetched by the BIU.

BIU and EU functions separately.

Bus Interface Unit (BIU)

BIU fetches instructions, reads data from memory and I/O ports, writes data to memory and I/ O ports.

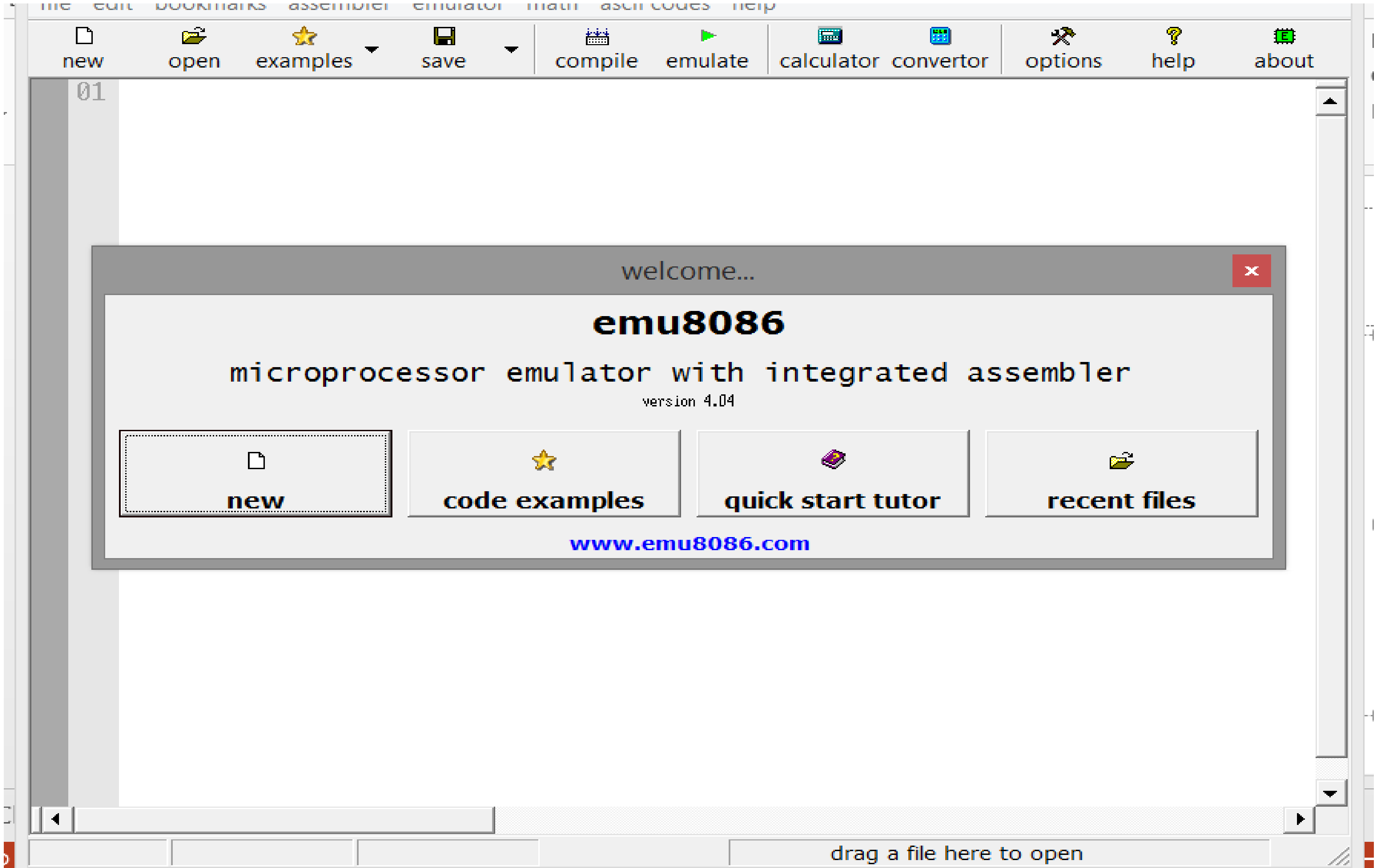
Register names

- Accumulator
- Base index
- Count
- Data
- Stack Pointer
- Base Pointer
- Destination index
- Source index
- Instruction Pointer
- Flags

Segment registers

- Code
- Data
- Extra
- Stack

Emulator 8086

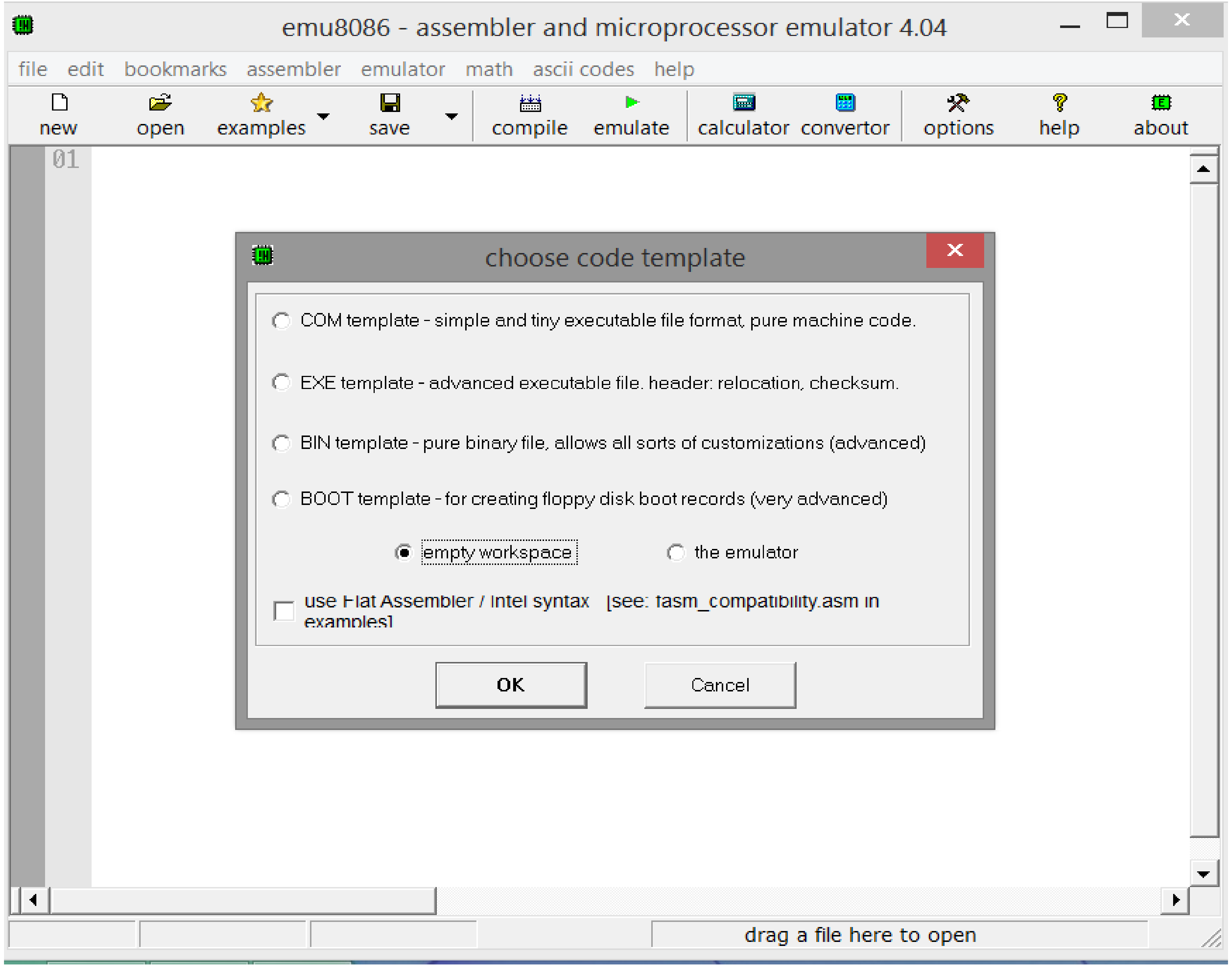


emu8086

microprocessor emulator with integrated assembler

version 4.04

www.emu8086.com


emu8086 - assembler and microprocessor emulator 4.04
file edit bookmarks assembler emulator math ascii codes help
new open examples save compile emulate calculator convertor options help about
drag a file here to open

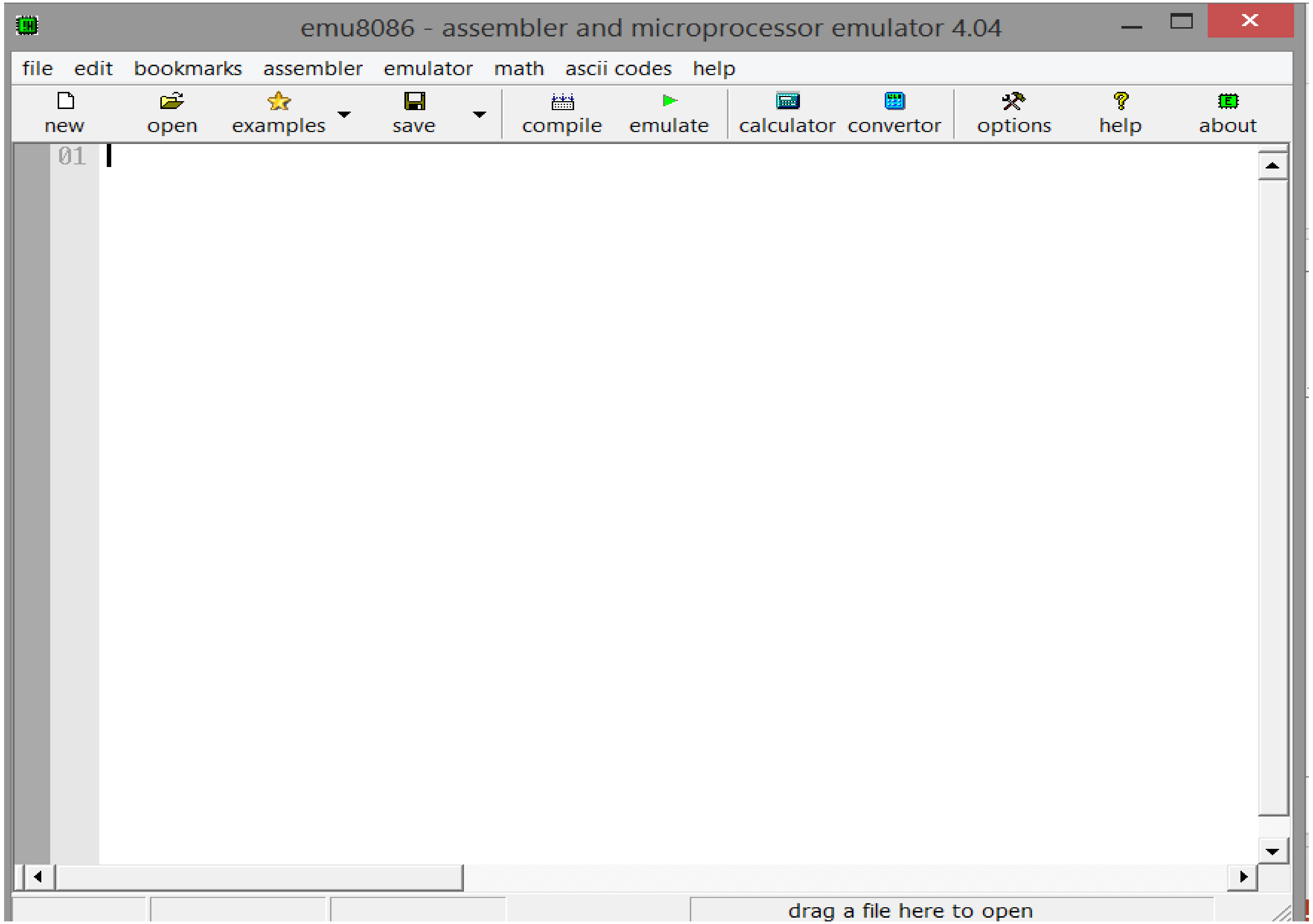
choose code template

- COM template - simple and tiny executable file format, pure machine code.
- EXE template - advanced executable file. header: relocation, checksum.
- BIN template - pure binary file, allows all sorts of customizations (advanced)
- BOOT template - for creating floppy disk boot records (very advanced)

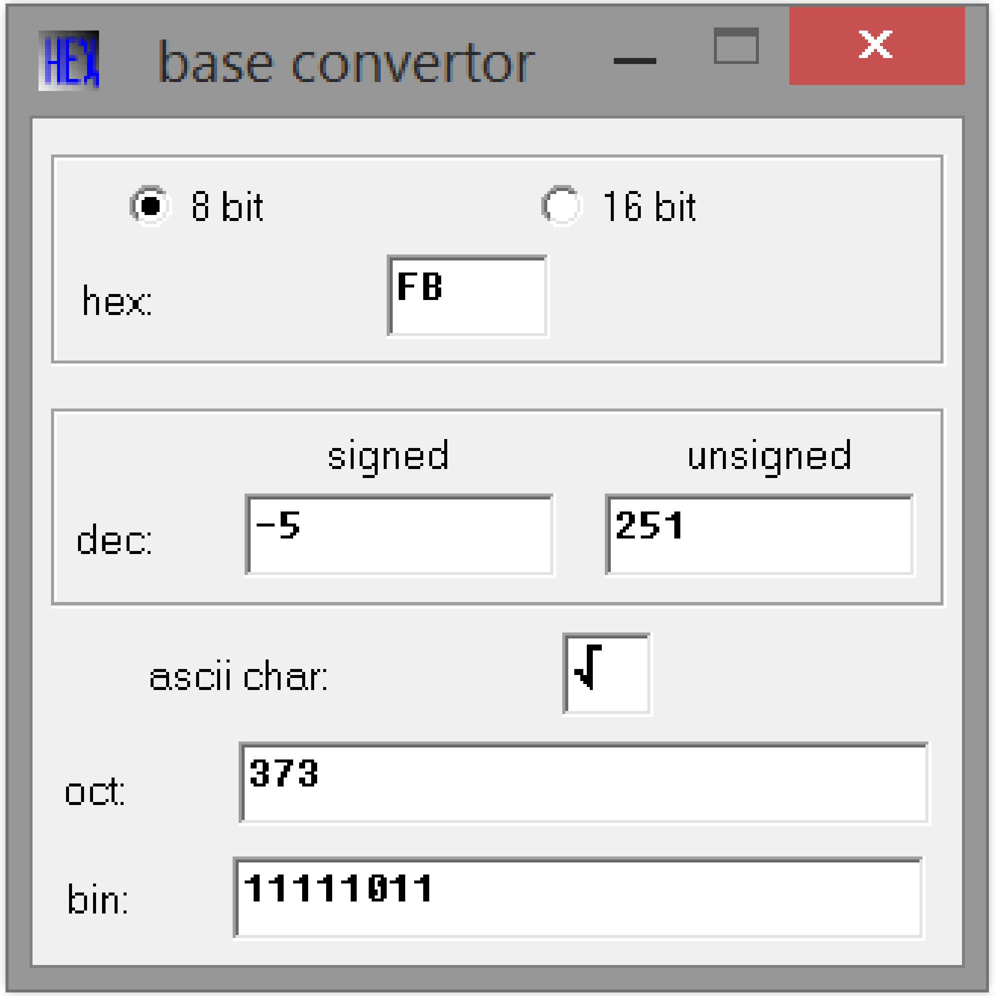
[✓ empty workspace] the emulator

☐ use Flat Assembler / Intel syntax [see: fasm_compatibility.asm in examples]

OK Cancel


emu8086 - assembler and microprocessor emulator 4.04
file edit bookmarks assembler emulator math ascii codes help
new
open
examples
save
compile
emulate
calculator
convertor
options
help
about
01
drag a file here to open

Signed Numbers



base convertor

[✓ 8 bit] 16 bit

hex: FB

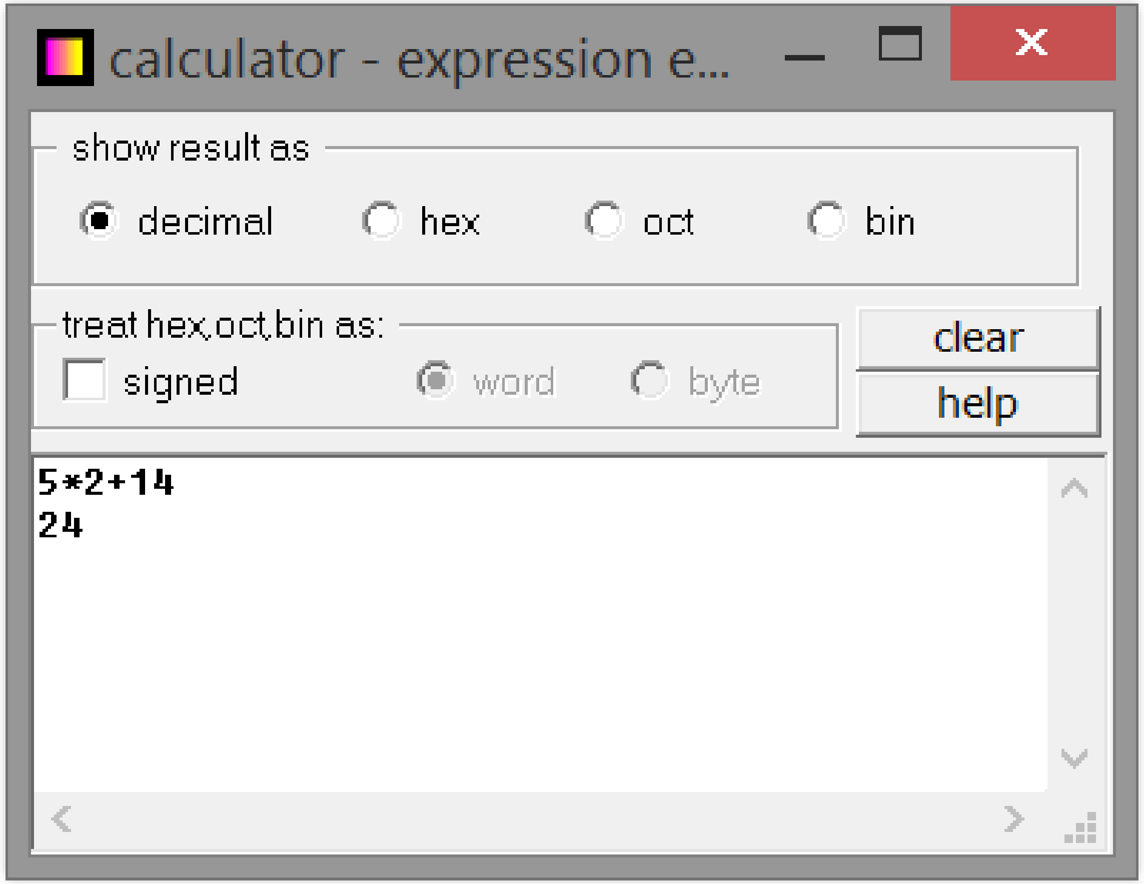
	signed	unsigned
dec:	-5	251

ascii char: √

oct: 373

bin: 11111011

Emulator Calculator


calculator - expression e...

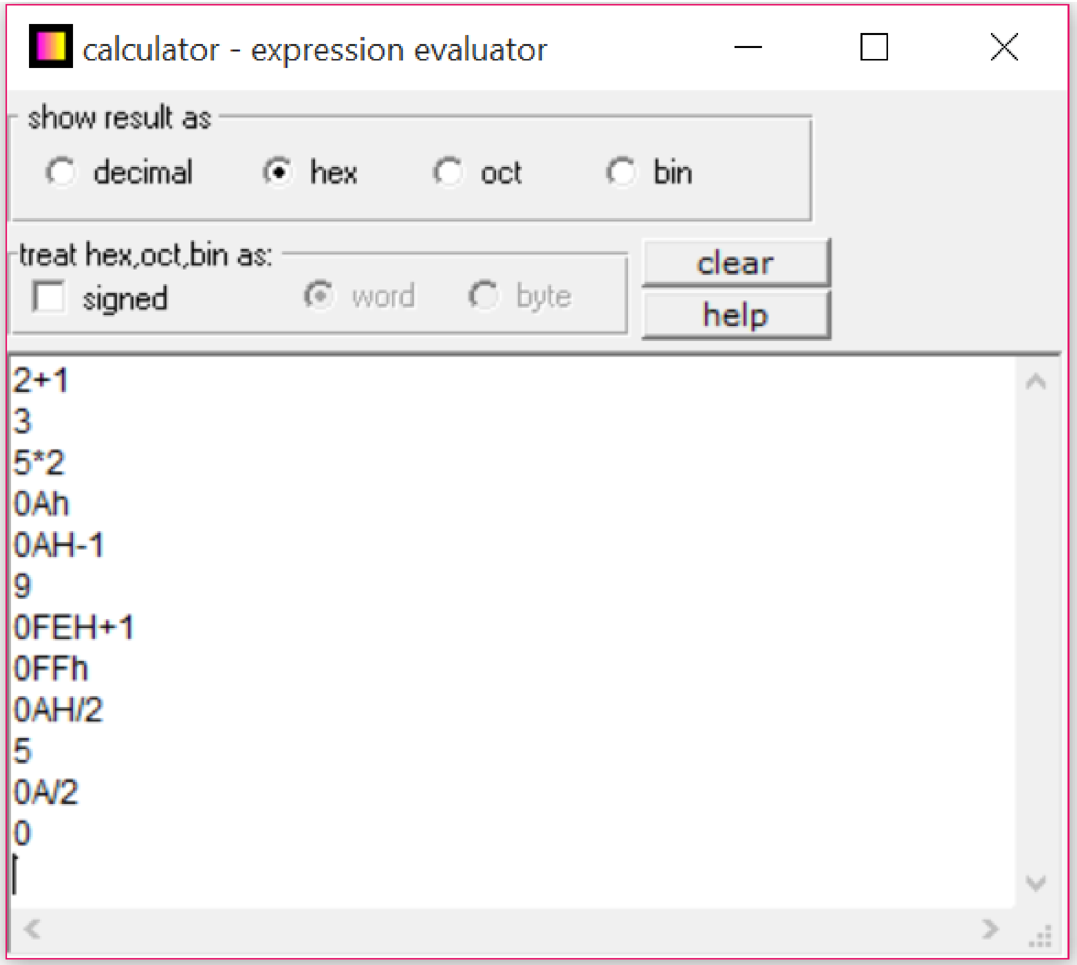
show result as

[✓ decimal] hex oct bin

treat hex,oct,bin as:

☐ signed [✓ word] byte

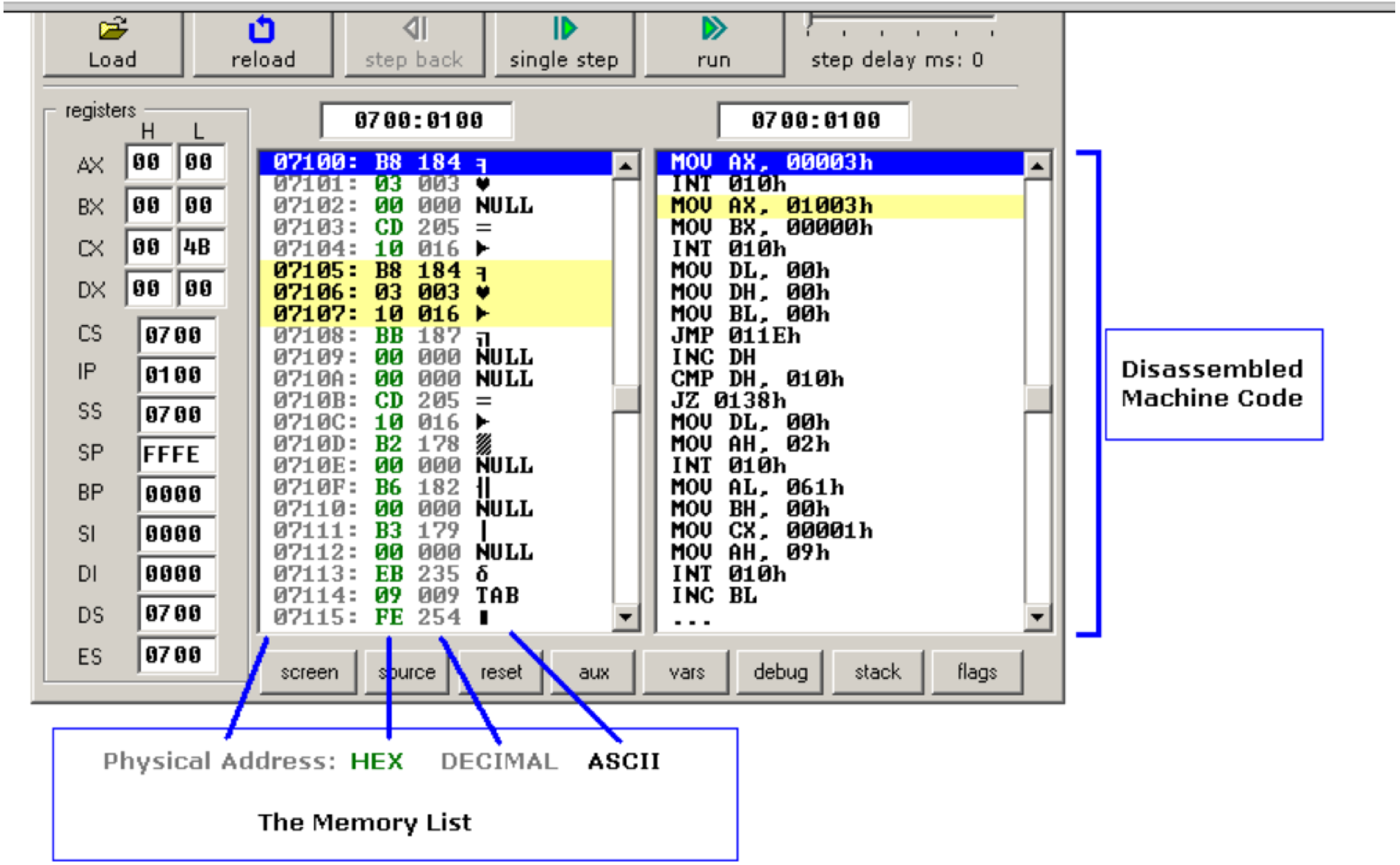
```
5*2+14
24
```



show result as: ( ) decimal [✓ hex] ( ) oct ( ) bin

treat hex,oct,bin as: ☐ signed [✓ word] ( ) byte

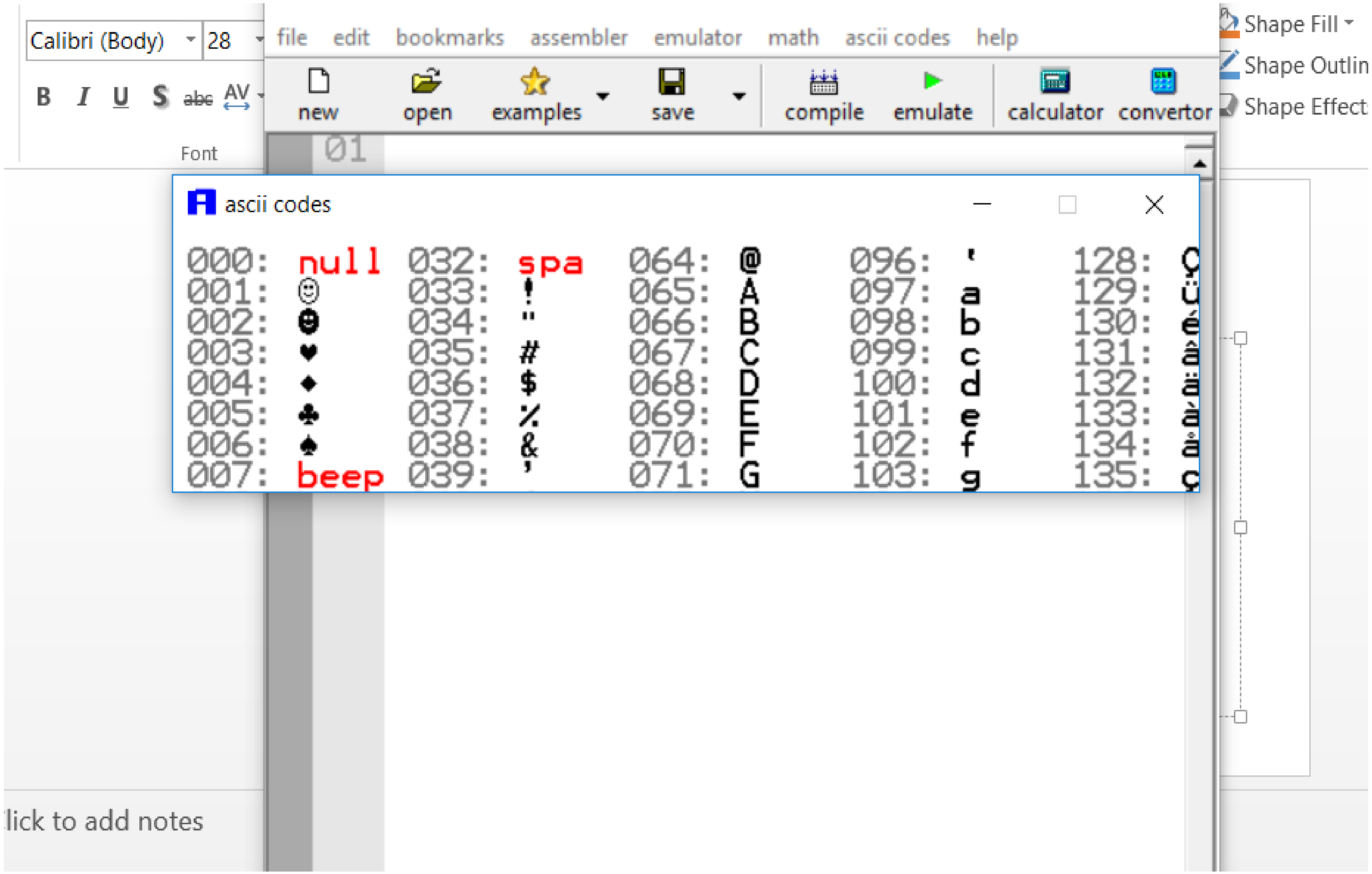
```
2+1
3
5*2
0Ah
0AH-1
9
0FEH+1
0FFh
0AH/2
5
0A/2
0
```



```
registers
     H   L
AX   00  00
BX   00  00
CX   00  4B
DX   00  00
CS   0700
IP   0100
SS   0700
SP   FFFE
BP   0000
SI   0000
DI   0000
DS   0700
ES   0700
```

```
0700:0100

07100: B8 184 ╕
07101: 03 003 ♥
07102: 00 000 NULL
07103: CD 205 ═
07104: 10 016 ►
07105: B8 184 ╕
07106: 03 003 ♥
07107: 10 016 ►
07108: BB 187 ╗
07109: 00 000 NULL
0710A: 00 000 NULL
0710B: CD 205 ═
0710C: 10 016 ►
0710D: B2 178 ▓
0710E: 00 000 NULL
0710F: B6 182 ╢
07110: 00 000 NULL
07111: B3 179 │
07112: 00 000 NULL
07113: EB 235 δ
07114: 09 009 TAB
07115: FE 254 ■
```

```
0700:0100

MOV AX, 00003h
INT 010h
MOV AX, 01003h
MOV BX, 00000h
INT 010h
MOV DL, 00h
MOV DH, 00h
MOV BL, 00h
JMP 011Eh
INC DH
CMP DH, 010h
JZ 0138h
MOV DL, 00h
MOV AH, 02h
INT 010h
MOV AL, 061h
MOV BH, 00h
MOV CX, 00001h
MOV AH, 09h
INT 010h
INC BL
...
```



```
000: null 032: spa 064: @ 096: `  128: Ç
001: ☺    033: !   065: A 097: a  129: ü
002: ☻    034: "   066: B 098: b  130: é
003: ♥    035: #   067: C 099: c  131: â
004: ♦    036: $   068: D 100: d  132: ä
005: ♣    037: %   069: E 101: e  133: à
006: ♠    038: &   070: F 102: f  134: å
007: beep 039: '   071: G 103: g  135: ç
```