Piotr Heinzelman

a) 1111 1111 (2) =

(16) 0xFFh

(8) 0377

(10) 255 - od razu widać :)

---

b) 1100 0011 (2) =

011.000.011

3 0 3 (8) = 0303 (8)

0xC3h (16) =

(10) = 0xC3(16) = "C"·16+3 = 12·16+3 =

= 128+36+36+3 = 195 (10)

---

c) 0001.0001 (2)

(8) → 00.010.001 = 021 (8)

(16) → 0x11h = 0x11h (16)

(10) → 0x11 → "1"·16 + "1" = 16+1 = 17 (10)

---

d) 0011.1100 (2)

(8) 00.111.100 = 074 (8)

(16) 0011.1100 = 0x3Ch

(10) 0x3Ch → (16) = "3"·16 + "C" = 48 + 12 = 60 (10)

---

algorytm konwersji 3 cyfrowego hex na kolejny oct. (ja we wzory oct.) ABC - Hex defg - oct.

ABC = xxxx.xxxx.xxxx (2) =

= defg xxx.xxx.xxx.xxx (8)

16: 0000 - 0
0001 - 1
0010 - 2
0011 - 3
0100 - 4
0101 - 5
0110 - 6
0111 - 7
1000 - 8
1001 - 9
1010 - A
1011 - B
1100 - C
1101 - D
1110 - E
1111 - F

8: 000 - 0
001 - 1
010 - 2
011 - 3
100 - 4
101 - 5
110 - 6
111 - 7

BX BX+1 BX+2

liczba "16" BCD = 567

liczba "8" efgh = ?

+9 +10 +11 +12

1) Wczytaj najmłodszą "16" D do rejestru AH

MOV AH, [BX+2] A: ....|xxxx|....|....

AH AL

2) Przesuń AX o 3 bity w prawo

(chyba że wolno dzielić przez 8)

MOV CL, 03;

SHR AX, CL; A: ....|...x|xxx.|....

3) Przesuń AL o 5 bitów w prawo

MOV CL, 05;

SHR AL, CL A: ....|...x|....|.xxx h = 7

AH AL

4) Zapisz najmłodszą część wyniku

MOV [BX+12], AL;

---

5) Przesuń AX o 1 bit w prawo

SHR AX, 1 A: ....|....|x...|..xx

6) Załaduj kolejną część do AH

MOV AH, [BX+1] A: ....|xxxx|x...|..x?

7) Przesuń AX o 2 bity w prawo

SHR AX, 1; SHR AX, 1; A: ....|..xx|xxx.|....

AH AC

8) Przesuń AL o 5 bitów w prawo

SHR AL, CL A: ....|..xx|....|.xxx

9) zapisz wynik do drugiego znaku "8" (4)

g = AL

MOV [BX+11], AL A: ····|··xx|····|·xxx

10) przesuń A o 2 bity w prawo

SHR AX,1; SHR AX,1;

A: ····|····|xx··|···x

11) wpisz 3 znak "16" do AH

MOV AH, [BX]

A: ····|xxxx|xx··|···x

12) przesuń AX o 1 bit w prawo

SHR AX,1 A: ····|xxx·|xxx·|····

13) przesuń AL w prawo o 5 bitów

SHR AL, CL A: ····|xxx·|····|·xxx

14) zapisz wynik AL w liczbie f

MOV [BX+10], AL

15) zapisz liczbę e

MOV [BX+9], AH

---

rozwiązanie w C

```
char B; char C; char D;
char d; char e; char f; char g;
int sum= (((B*16)+C)*16)+D; // *16 to to samo co <<4 lub SHR
g=sum%8;
sum=sum>>3;  // sum>>3 to to samo co sum/8 tylko szybciej.
f=sum%8;
sum=sum/8;
e=sum%8;
d=sum>>3;
```

zamiast "/" można użyć A >> 3;

algorytm jest podobny i polega na wyliczaniu wartości przez mnożenie liczby przez wagę, a następnie obliczenie kolejnych znaków od prawej storny przez dzielenie wartości z resztą.

tutorialspoint | Online Assembly Compiler

Execute | Beautify | Share | Source Code | Help | Login ×

```
section .text
    global _start       ;must be declared for using gcc
_start:                     ;tell linker entry point
    mov ah,[D]
    mov CL,3
    SHR AX,CL
    MOV CL,05
    SHR AL,CL
    ADD AL,0x30 ; zamiana na ascii przez dodanie x30
    MOV [h],AL
    SHR AX,1;
    MOV AH,[C]
    SHR AX,1
    SHR AX,1
    SHR AL,CL
    ADD AL,0x30 ; zamiana na ascii
    MOV [g],AL
    SHR AX,1
    SHR AX,1
    MOV AH,[B]
    SHR AX,1
    SHR AL,CL
    ADD AL,0x30 ; ...
    MOV [f],AL
    ADD AH,0x30
    MOV [e],AH
    MOV AX,[B]
    ADD AX,0x30 ; zmiana na ascii
    MOV [B],AX
    MOV AX,[C]
    ADD AX,0x30
    MOv [C],AX
    MOV AX,[D]
    ADD AX,0x30
    MOv [D],AX


    mov edx, len    ;message length
    mov ecx, B    ;message to write
    mov ebx, 1      ;file descriptor (stdout)
    mov eax, 4      ;system call number (sys_write)
    int 0x80        ;call kernel
    mov eax, 1      ;system call number (sys_exit)
    int 0x80        ;call kernel

section .data
B db 5
C db 6
D db 7
xx db 0x3d
e db 0
f db 0
g db 0
h db 0


msg db  'Hello, world!',0xa ;our dear string
len equ $ - msg         ;length of our dear string
```

Terminal

```
567=2547Hello,
```

Dlaczego heksadecymalny?
Proste - aby wyświetlić wartości liczbowe zapisane w pamięci musimy je wielokrotnie dzielić 2 razy przez podstawę, czyli dla liczb "zwykłych" przez 10 - co jest kłopotliwe dla maszyn przetwarzających liczby o podstawie 2.
Natomiast dzielenie przez 2, 4, 8, 16 i kolejne potęgi liczby 2 polega na przesuwaniu bitów w prawo o 1, 2, 3 lub 4 pozycje.
Łatwiej jest wyświetlić liczbę przy podstawie 8 lub 16. Podstawa 16 ma tę zaletę że ma zawsze 2 znaki dla każdych 8 bitów, podstawa 8 może mieć 2 lub 3 znaki.
Pozostaje zapis heksadecymalny od razu wiedzieć jeśli pojawi się znak "10" A "11" B lub podobne.
Zapis 16 jest po prostu wygodny. A poza tym jest trochę elitarny, co ma także znaczenie przy powstaniu HACK-mowy czyli zapisu słów z wykorzystaniem cyfr i znaków. Dobre to właśnie do konstruowania haseł np: "h@sło" zamiast "hasło".

Zadanie 3

		x16
0	....	0
1	...1	16
2	..1.	32
3	..11	48
4	.1..	64
5	.1.1	80
6	.11.	96
7	.111	112
8	1...	128
9	1..1	144
A	1.1.	160
B	1.11	176
C	11..	192
D	11.1	208
E	111.	224
F	1111	240

$-58_{(10)} \rightarrow$

$58_{(10)} = 48 + 10 = 0x30 + A = 0x3A$

$0x3A = ..11.1.1.$

$-0x3A = 11...1.1$

$= 0xC5$

$-58_{(u1)} = 0xC5 = 0xFFC5$

$0x003A$

$0x0000003A$

$0xFFFF$

$FFC5$

$-128_{(10)} \rightarrow 128_{(10)} = 0x80+0 = 0x80 = 1.....$

$-128_{(u1)} = .111.1111 = 0x7F$

$-128_{(u1)} = 0xFF7F \;/\; 0xFFFF\,FF7F$

$-1023_{(10)}\ u_1 \rightarrow$ ($1024 = 2^{10}$ $1023 = 0011.1111.1111$

$0x3F.F$

$-1023 = 0xFC00$

$-1023_{(u1)} = 0xFC00 \;/\; 0xFFFF\,FC00$

Zadanie 4

$-58\ u_2 = -58\,u_1 + 1 = 0xFFC5 + 1 = 0xFFC6$

$0xFFFF\,FFC5 + 1 = 0xFFFF\,FFC6$

$58\ u_2 = 58\,u_1 = 0x3A = 0x003A$

$0x0000\,003A$

$-128_{(u2)} = -128_{(u1)} + 1 = 0xFF7F + 1 = 0xFF80$

$= 0xFFFF\,FF80$

$-1023_{(u2)} = -1023_{(u1)} + 1 =$

$0xFC00 + 1 = 0xFC01$

$0xFFFF\,FC01$

Zad 5 Czy dodanie liczb 1111 1111 da poprawny wynik?
Zobaczmy:

1111 1111 to (1)111 1111
− 111 1111 −1
− 111 1110 odejmujemy
− 0000 0001

= −1 (10)

0 ... 0010	2
0 ... 0001	1
0 ... 0000	0
1 ... 1111	−1
1 ... 1110	−2
1 ... 1101	−3

−1 + −1 = −2

Suma sumatora

```
  1111 1111
+ 1111 1111
-----------
(1)1111 1110
```

= 1111 1110
= −111 1110 (−1)
−111 1101 odejmujemy
−000 0010

= −2

Wynik jest prawidłowy, −1 + (−1) = −2.

Uzupełnienie do 2 jest najpopularniejsze, ponieważ daje poprawne wyniki przy dodawaniu (odejmowaniu liczb dodatnich i ujemnych, o ile nie przekroczymy zakresu.

Zad 5a czy dodawanie liczb 1000 0000 i 1000000 da poprawny wynik?

1000 0000 U2 = (-) 000 0000 => (-1) = (-)0 111 1111 odwracamy

- 10000000 czyli jest to liczba -128.

dodawanie dwóch liczb -128 + (-128) da w wyniku -256. czyli wynik nie zmieści się zakresie 8 bitowym.

$$\begin{array}{r} 10000000 \\ + \ 10000000 \\ \hline (1)\ 00000000 \end{array} = 0$$

wynik nie jest prawidłowy, ustawione prawdopodobnie flagi przeniesienia i

CF OF
+ —

w 8086 ustawione Carry flag CY +
NIE ustawione Overflow flag NV -

OF + CF +

Przy dodawaniu 100000 flagi OV CY

Zad 6

liczba pierwsza 1000 1000 - spakowana
liczba druga 0001 1000 - spakowana
dodawane liczb BCD.

0	....
1	...1
2	..1.
3	..11
4	.1..
5	.1.1
6	.11.
7	.111
8	1...
9	1..1
A	1.1.
B	1.11
C	11..
D	11.1
E	111.
F	1111

1) Rozpakowanie drugą w BX

MOV BX, 0x18 ·····|xxxx yyyy
ROL BX, 4 ····xxxx|yyyy····
ROR BL, 4 ····xxxx|····yyyy

0001 1000 = 18
1000 1000 = 88

Rozpakowujemy - czyli rozkładamy 8 bitów na 16 bitów.
To samo liczba pierwsza

MOV AX, 88 ·····|xxxx yyyy
ROL AX, 4 ····xxxx|yyyy····
ROR BL, 4 ····xxxx|····yyyy

```
  0808
+ 0108
------
  0910
```

ustawione flaga AF

2) dodajemy

```
  ····aaaa|····bbbb
+ ····cccc|····dddd
------------------
  ····ffff  ···1eeee
```

AF (+)

jeśli następuje przeniesienie na 4 bicie to musimy wynik „wyrównać" ponieważ nie jest on zgodny z kodem BCD.

3) jeśli AF to:

AAA - wyrównanie po dodawaniu BCD.

0910 → AAA → AH+1, AL+6 + maska na wcześniejsze bity
→ 1006 → ~~AAA~~ mamy przeniesienie
→ (1).06 wynik - dodajemy 0x30 do każdej liczby wyniku by dostać kod ASCII i wyświetlić

Zad 6a)

DX AX + 0x30

0 1 0 6

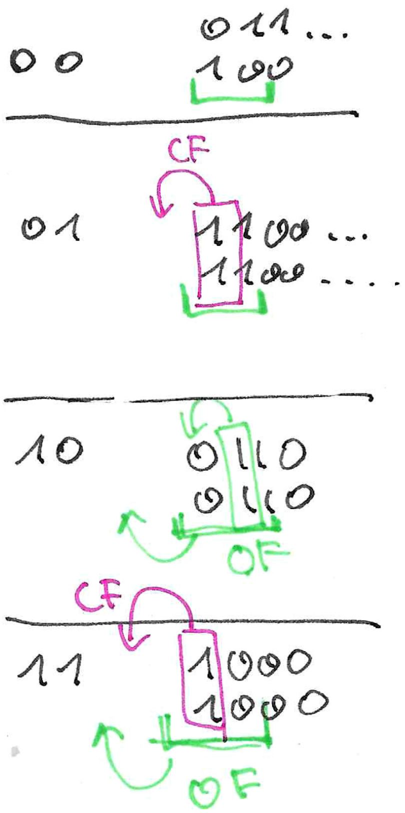
+ 30 30 30 30

---

30 31 30 36 = ASCII "0106" znaki.

Zad 7) czy możliwe jest przy dodawaniu liczb 8-bit. uzyskanie wszystkich układów flag

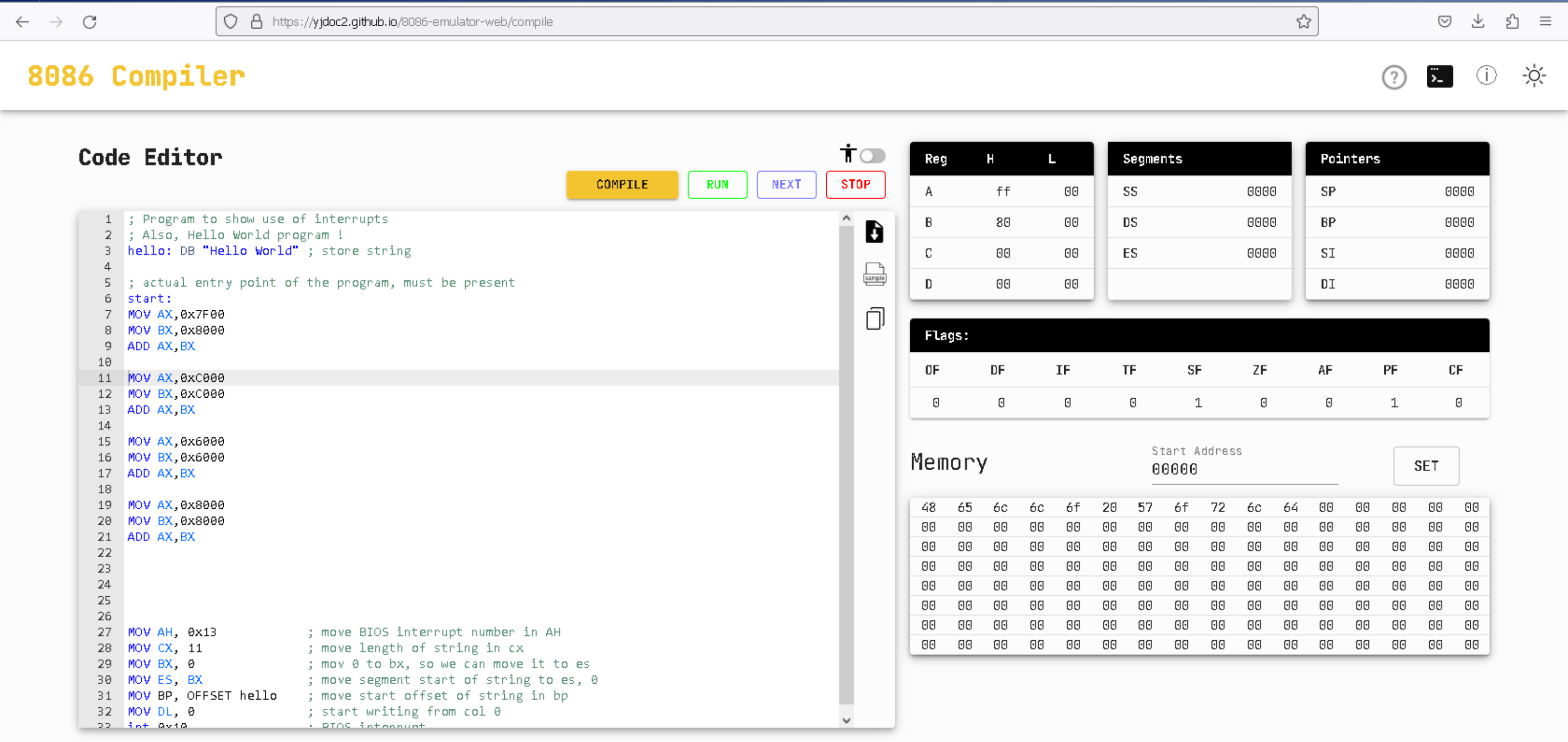
OF Overflow Flag	Carry Flag CF	
0	0	0x7F, 0x80 (0111 1111, 1000 0000)
0	1	0xC0, 0xC0 (1100 0000)
1	0	0x60, 0x60 (0110 0000)
1	1	0x80, 0x80 (1000 0000)

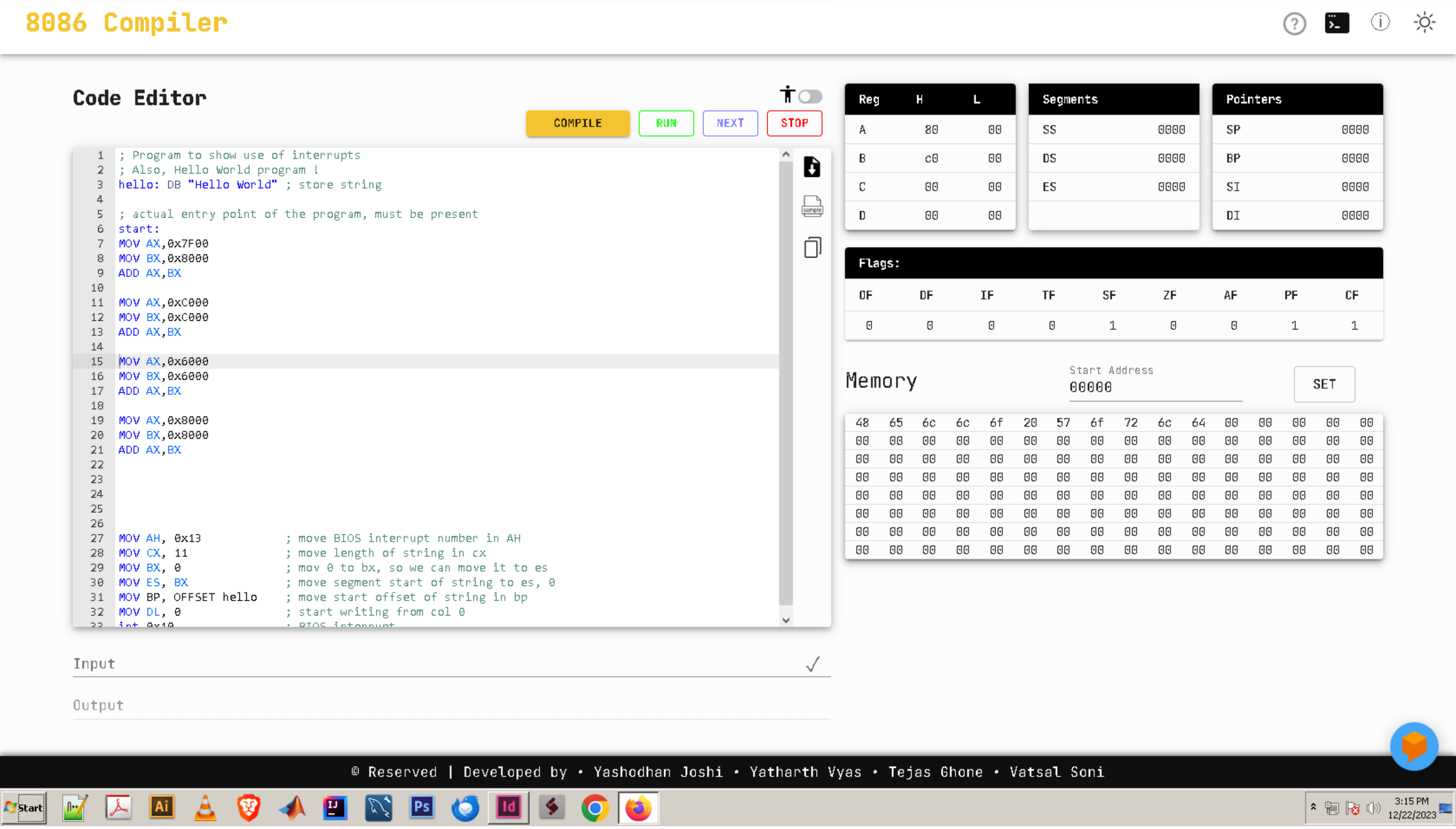


CF=1 – ostatni bit ma przeniesienie

OF=1 – jeśli 7 i 8 mają przeniesienie

$(7+8)\%2=1$


https://yjdoc2.github.io/8086-emulator-web/compile
8086 Compiler
Code Editor
COMPILE
RUN
NEXT
STOP
; 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 AX,0x7F00
MOV BX,0x8000
ADD AX,BX
MOV AX,0xC000
MOV BX,0xC000
ADD AX,BX
MOV AX,0x6000
MOV BX,0x6000
ADD AX,BX
MOV AX,0x8000
MOV BX,0x8000
ADD AX,BX
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
Reg H L
A ff 00
B 80 00
C 00 00
D 00 00
Segments
SS 0000
DS 0000
ES 0000
Pointers
SP 0000
BP 0000
SI 0000
DI 0000
Flags:
OF DF IF TF SF ZF AF PF CF
0 0 0 0 1 0 0 1 0
Memory
Start Address
00000
SET
48 65 6c 6c 6f 20 57 6f 72 6c 64 00 00 00 00 00


8086 Compiler
Code Editor
COMPILE
RUN
NEXT
STOP
; 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 AX,0x7F00
MOV BX,0x8000
ADD AX,BX
MOV AX,0xC000
MOV BX,0xC000
ADD AX,BX
MOV AX,0x6000
MOV BX,0x6000
ADD AX,BX
MOV AX,0x8000
MOV BX,0x8000
ADD AX,BX
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
Reg H L
A 80 00
B c0 00
C 00 00
D 00 00
Segments
SS 0000
DS 0000
ES 0000
Pointers
SP 0000
BP 0000
SI 0000
DI 0000
Flags:
OF DF IF TF SF ZF AF PF CF
0 0 0 0 1 0 0 1 1
Memory
Start Address
00000
SET
48 65 6c 6c 6f 20 57 6f 72 6c 64 00 00 00 00 00
Input
Output
© Reserved | Developed by • Yashodhan Joshi • Yatharth Vyas • Tejas Ghone • Vatsal Soni
3:15 PM
12/22/2023

https://yjdoc2.github.io/8086-emulator-web/compile

8086 Compiler

Code Editor

COMPILE | RUN | NEXT | STOP

```
; 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 AX,0x7F00
MOV BX,0x8000
ADD AX,BX

MOV AX,0xC000
MOV BX,0xC000
ADD AX,BX

MOV AX,0x6000
MOV BX,0x6000
ADD AX,BX

MOV AX,0x8000
MOV BX,0x8000
ADD AX,BX





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
```

Reg	H	L
A	c0	00
B	60	00
C	00	00
D	00	00

Segments	
SS	0000
DS	0000
ES	0000

Pointers	
SP	0000
BP	0000
SI	0000
DI	0000

Flags:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
1	0	0	0	1	0	0	1	0

Memory

Start Address 00000 SET

48	65	6c	6c	6f	20	57	6f	72	6c	64	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Code Editor

COMPILE | RUN | NEXT | STOP

```
; 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 AX,0x7F00
MOV BX,0x8000
ADD AX,BX

MOV AX,0xC000
MOV BX,0xC000
ADD AX,BX

MOV AX,0x6000
MOV BX,0x6000
ADD AX,BX

MOV AX,0x8000
MOV BX,0x8000
ADD AX,BX





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
```

Reg	H	L
A	c0	00
B	60	00
C	00	00
D	00	00

Segments	
SS	0000
DS	0000
ES	0000

Pointers	
SP	0000
BP	0000
SI	0000
DI	0000

Flags:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
1	0	0	0	1	0	0	1	0

Memory

Start Address 00000 SET

48	65	6c	6c	6f	20	57	6f	72	6c	64	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Zad 8)

$11111111_{U2} = -1$ NBC

$10000000_{U2} = -128$ NBC

127	01111111
2	10
1	01
0	00
−1	1...1111
−2	1...1110
−128	10000000

Która jest większa?

w sensie || wartości bezwzględnej.

$|-128| > |-1|$

w sensie wartości

$-128 < -1$

Suma nie mieści się w zakresie, −128 jest ostatnią która mieści się w zapisie U2.

$U_2 < +127 \div -128 >$

2ad 9) zapis rextowy

539 (7, 11, 13, 17, 19)

DIV BX

$AX = (DX\ AX)/BX$

$DX = \%$

$539 \% 7 = 0$

$539 \% 11 = 0$

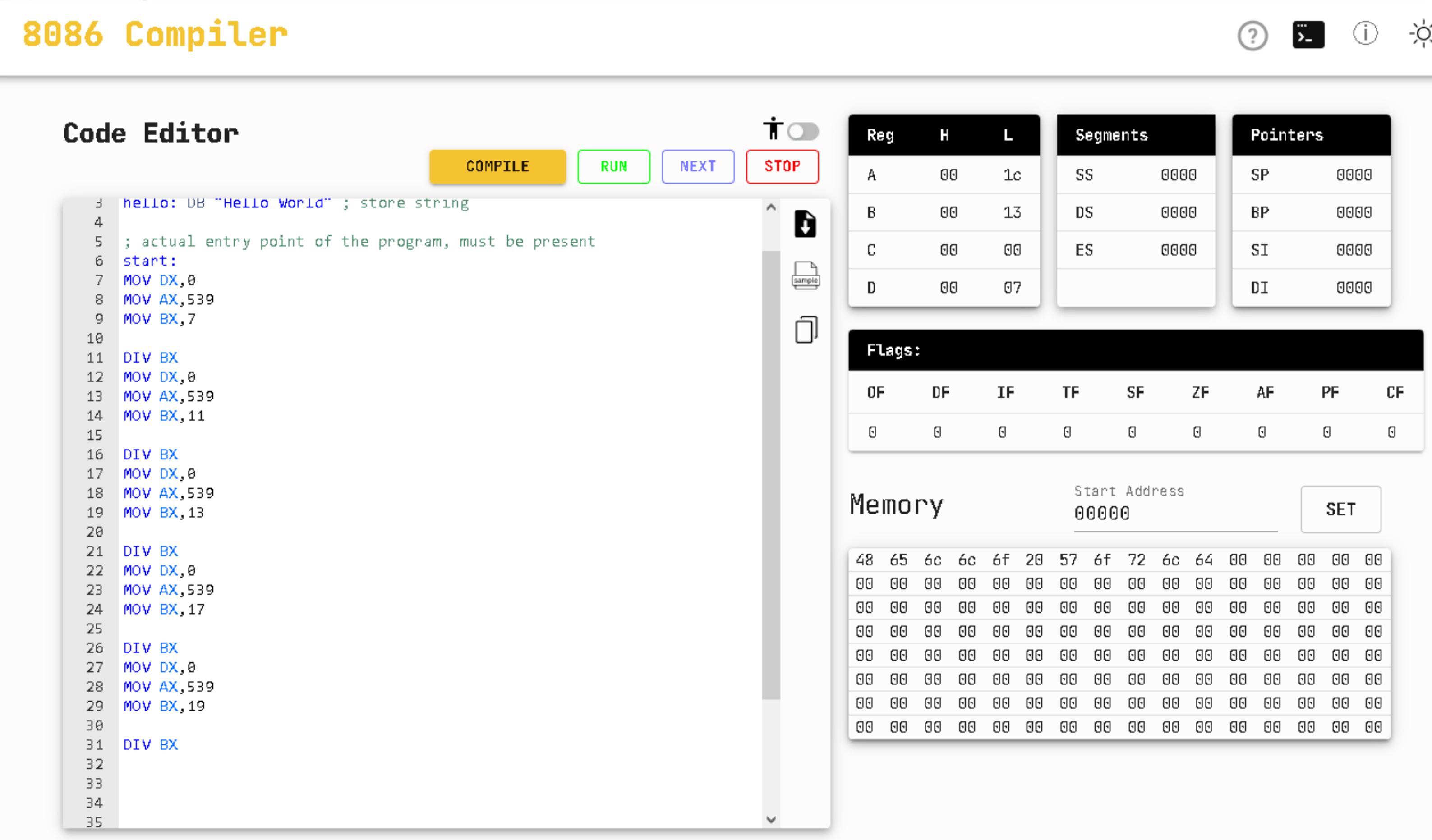
$539 \% 13 = 6$

$539 \% 17 = 0xC = 12$

$539 \% 19 = 0x7 = 7$

$m_1 = 7 \quad m_2 = 11 \quad m_3 = 13 \quad m_4 = 17 \quad m_5 = 19$

$539 = (0, 0, 6, 12, 7)$



8086 Compiler

Code Editor

COMPILE RUN NEXT STOP

```
hello: DB "Hello World" ; store string

; actual entry point of the program, must be present
start:
MOV DX,0
MOV AX,539
MOV BX,7

DIV BX
MOV DX,0
MOV AX,539
MOV BX,11

DIV BX
MOV DX,0
MOV AX,539
MOV BX,13

DIV BX
MOV DX,0
MOV AX,539
MOV BX,17

DIV BX
MOV DX,0
MOV AX,539
MOV BX,19

DIV BX
```

Reg	H	L
A	00	1c
B	00	13
C	00	00
D	00	07

Segments	
SS	0000
DS	0000
ES	0000

Pointers	
SP	0000
BP	0000
SI	0000
DI	0000

Flags:

OF	DF	IF	TF	SF	ZF	AF	PF	CF
0	0	0	0	0	0	0	0	0

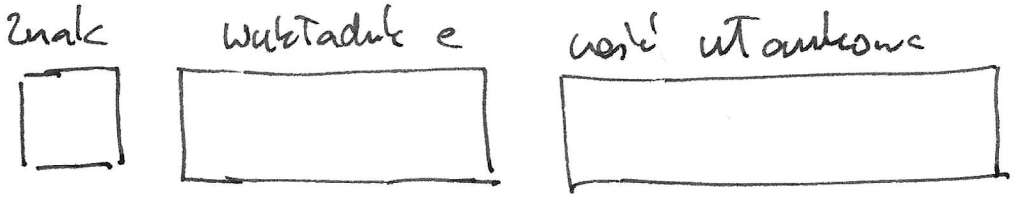
Memory

Start Address 00000 SET

48	65	6c	6c	6f	20	57	6f	72	6c	64	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

10

$129_{(10)}$ = IEEE (32 bit)



(0 1 11 12 63) zapis 64 Bit

1.f

$129_{(10)} = 00000010000001._{(2)}$ = ~~1.f~~

$= 1,0000001 \cdot 2^7$ = 1+f =>

f = 00000010000....

e = 7(10) = 7+127 = 128+6 =

= 1000.0110

e = 10000110

f = 00000010000000000000000

znak = 0 (dodatnie)

IEEE = 0 . 10000110 . 00000010000000000000000