

# Piotr Heinzelman

①

a)  $1111\ 1111_{(2)} =$

$(16) \text{ D} \times \text{FFh}$

$(8) \text{ 0377}$

$(10) \text{ 255 - od razy u mida :)}$

b)  $1100\ 0011_{(2)} =$

$$\begin{array}{r} 011.000.01 \\ 3 \quad 0 \quad 3 \end{array} \quad (8) = 0303_{(8)}$$

$0x C3h \quad (16) =$

$$\begin{aligned} (10) &= 0xC3_{(16)} = "C"\cdot 16 + 3 = 12\cdot 16 + 3 = \\ &= 120 + 36 + 3 = 195_{(10)} \end{aligned}$$

c)  $0001.0001_{(2)}$

$(8) \Rightarrow 00.010.001 = 021_{(8)}$

$(16) \Rightarrow 0x11h = 0x11h \quad (16)$

$(10) \Rightarrow 0x11 \rightarrow "1"\cdot 16 + "1" = 16 + 1 = 17_{(10)}$

d)  $0011.1100_{(2)}$

$(8) 00.111.100 = 074_{(8)}$

$(16) 0011.1100 = 0x3Ch$

$(10) 0x3Ch \rightarrow (10) = "3"\cdot 16 + "C" = 48 + 4^4 = 48 + 12 = 60_{(10)}$

algorytm konwersji 3 cyfrowego hex na liczbę oct.  
(ja w liczbach oct.) ABC - Hex defg - oct.

$ABC = \dots \dots \dots \dots \quad (2) =$

$= \text{defg} \quad \dots \dots \dots \dots \quad (8)$

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	...	- 0
	..1	- 1
	.1.	- 2
	.11	- 3
	1..	- 4
	1..1	- 5
	11.	- 6
	111	- 7

$Bx \quad BF^1 \quad BX+2$

(3)

liczba "16"  $BCD = 567$

liczba "8"  $efgh = ?$

+9 +10  
+4 +12

1) Wczytaj najmłodsze "16" do rejestrów AH  
 $MOV AH, [BX+2]$  A:  $\dots |xxxx| \dots | \dots$   
AH AL

2) Pniesź AX o 3 bity w prawo  
(czyścia 2e wokół dzielic pncz 8)

$MOV CL, 03;$  A:  $\dots | \dots x |xxx-1\dots$   
 $SHR AX, CL;$

3) Pniesź AL o 5 bitów w prawo

$MOV CL, 05;$  A:  $\dots | \dots x | \dots | \dots \star\star$  h=7  
~~SHR AX, CL~~ AH AL

4) Zapisz najmłodsze liczbę wynikową

$MOV [BX+12], AL;$

5) Pniesź AX o 1 bit w prawo

$SHR AX, 1$  A:  $\dots | \dots | x \dots | \dots xx$

6) Zataduj kolejną liczbę do AH

$MOV AH, [BX+1]$  A:  $\dots |xxxx| x \dots | \dots x \star$

7) Pniesź AX o 2 bity w prawo

$SHR AX, 1; SHR AX, 1;$  A:  $\dots | \dots xx | xxxx. | \dots$   
AH AL

8) Pniesź AL o 5 bitów w prawo

~~SHR AL, CL~~ A:  $\dots | \dots xx | \dots | \dots \star\star$

9) Zapisz wynik do drugiego znaku "8"  
 $g = AL$

MOV [BX+4], AL      A: ...|..xx|...|..xx

10) Przesuń A o 2 bity w prawo  
SHR AX, 1; SHR AX, 1;

A: ....|....|xx..|...x

11) Wątpliwy 3 znak 16<sup>4</sup> 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|.....|...xx

14) Zapisz wynik fC w liczbę f

MOV [BX+4], AL

15) Zapisz litery 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 samo co <<4 lub SHR  
g=sum%8;  
sum=sum>>3; // sum>>3 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.

[Execute](#) | [Beautify](#) | [Share](#) [Source Code](#) [Help](#) [Login](#)

Terminal

```
1 section .text
2     global _start          ;must be declared for using gcc
3 _start:
4     mov ah,[D]
5     mov CL,3
6     SHR AX,CL
7     MOV CL,05
8     SHR AL,CL
9     ADD AL,0x30 ; zamiana na ascii przez dodanie x30
10    MOV [h],AL
11    SHR AX,1;
12    MOV AH,[C]
13    SHR AX,1
14    SHR AX,1
15    SHR AL,CL
16    ADD AL,0x30 ; zamiana na ascii
17    MOV [g],AL
18    SHR AX,1
19    SHR AX,1
20    MOV AH,[B]
21    SHR AX,1
22    SHR AL,CL
23    ADD AL,0x30 ; ...
24    MOV [f],AL
25    ADD AH,0x30
26    MOV [e],AH
27    MOV AX,[B]
28    ADD AX,0x30 ; zmiana na ascii
29    MOV [B],AX
30    MOV AX,[C]
31    ADD AX,0x30
32    MOV [C],AX
33    MOV AX,[D]
34    ADD AX,0x30
35    MOV [D],AX
36
37
38    mov edx, len      ;message length
39    mov ecx, B        ;message to write
40    mov ebx, 1        ;file descriptor (stdout)
41    mov eax, 4        ;system call number (sys_write)
42    int 0x80          ;call kernel
43    mov eax, 1        ;system call number (sys_exit)
44    int 0x80          ;call kernel
45
46 section .data
47 B db 5
48 C db 6
49 D db 7
50 xx db 0xd
51 e db 0
52 f db 0
53 g db 0
54 h db 0
55
56
57 msg db 'Hello, world!',0xa ;our dear string
58 len equ $ - msg           ;length of our dear string
```

567=2547Hello,

① Placego hexadecymalny?

proste - aby mielić waroty lubiące zapiąć w  
pawie masingi iż wielokrotnie dwie  
2 rentę przed podstawną, aylej dle tych  
u "założyci" pier 10 - co jest konieczne dla  
maszyn przetwarzających lecaby o podstawie 2.

Natomiast dwie pier 2, 4, 8, 16 i lecaby  
podstawić lecaby 2 podlega na presowane bieżniki  
w prawo o 1, 2, 3 lub 4 pary.

Zauważ jasne wykrocie lecaby podstawną 8  
lub 16. podstawną 16 ma tg założycie  
ma 2ane 2 male dla każdej 8 bieżniki,  
podstawną 8 more nai 2 lub 3 zuchi'.

Ponadto zapiąć hexadecymalny od nowa mialce'  
jeśli pojawi się zuch "10" A "11" B lub niektóre.

zapiąć 16 jest po prostu wygodniej. A poza tym  
jest trochę delikatniej, co ma duże znaczenie  
my ponadto HICK-NOWY aylej zapin skróć  
2 ujemnikiem cyfr i masingi. Bywa to  
wymuszone do konstrukcji kaset np: "h@sto"  
zamiast "hasto".

## Zadanie 3

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

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

$$0x3A = \cdot\cdot\cdot\cdot\cdot\cdot\cdot$$

$$\begin{array}{l} 0x003A \\ 0x0000003A \end{array}$$

$$\underline{-58_{(u_1)}} - \underline{0x3A} = \underline{\underline{11\cdots11}} = \underline{0xC5} = 0xFFC5$$

$$= 0xC5$$

$$\underline{0xFFFF}$$

$$\underline{FFC5}$$

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

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

$$-128_{(u_1)} = \underline{\underline{\cdots\cdots\cdots\cdots}} = \underline{\underline{1111.1111}} = 0x7F$$

$$-128_{(u_1)} = 0xFF7F / 0xFFFF FF7F$$

$$-1023_{(10)} u_1 \rightarrow (1024 = 2^{10} 1023 = 0011.1111.1111 \\ 0x3 F.F)$$

$$-1023 = \underline{\underline{0xF C 00}}$$

$$-1023_{(u_1)} = 0xFC00 / 0xFFFF FC00$$

$$Zadanie 4 \quad -58_{u_2} = -58_{u_1} + 1 = 0xFFC5 + 1 = \underline{\underline{0xFFC6}}$$

$$0xFFFF FFC5 + 1 = \underline{\underline{0xFFFF FFC6}}$$

$$58_{u_2} = 58_{u_1} = 0x3A = \underline{\underline{0x003A}}$$

$$\underline{\underline{0x0000 003A}}$$

$$-128_{(u_2)} = -128_{(u_1)} + 1 = 0xFF7F + 1 = \underline{\underline{0xFF80}}$$

$$= \underline{\underline{0xFFFF FF80}}$$

$$-1023_{(u_2)} = -1023_{(u_1)} + 1 =$$

$$0xFC00 + 1 = \underline{\underline{0xFC01}}$$

$$\underline{\underline{0xFFFF FC01}}$$

Zad 5 Czy dodanie liczb 1111 1111 da poprawny wynik?  
Zobacz.

$$\begin{array}{r}
 11111111 \text{ to } (1)1111111 \\
 -1111111 \quad -1 \\
 -1111110 \quad \text{obracaj} \\
 -0000001 \\
 = -1(10) \\
 -1 + -1 = \boxed{-2}
 \end{array}$$

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

Suma sumatora

$$\begin{array}{r}
 \begin{array}{r}
 11111111 \\
 11111111 \\
 + 11111111 \\
 \hline (0)11111110
 \end{array} \\
 = 1111110 \\
 = -11111110 (-1) \\
 -11111101 \quad \text{obracaj} \\
 -00000010 \\
 = -2
 \end{array}$$

Wynik jest prawidły,  $-1 + (-1) = -2$ .

Uzupelnienie do 2 jest najpopularniejsze, poniewaz  
dalej poprawie wyników przy dodawaniu (odejmowaniu) liczb  
dodatnich i ujemnych, o ile nie przekroczymy  
zakresu.

2ad 5a my dodaje liczb 1000 0000 i 10000000 da  
poprawny wynik?

$$1000 \ 0000 \cup_2 = (-)000 \ 0000 \Rightarrow (-1) = (-)0 \ 111\ 111 \text{ odwzor.}$$

- 10000000 cyfli jest do  
liczby -128.

dodaję dalej liczb -128 + (-128) da w  
wyniku -256. cyfli wynik nie zmieni się  
zakroć 8bitów wyn.

$$\begin{array}{r} 1000 \ 0000 \\ + 1000 \ 0000 \\ \hline 0 \ 000 \ 0000 \end{array} = 0 \quad \text{wynik nie jest prawidłowy,}  
ustawione pseudo-podobne flagi:  
przerwy:$$

CF      OF  
+      —

wc 8086 ustawione Carry Flag CY +  
NIE ustawione Overflow Flag NV -

---

my dodawałem 100000 flagi OV CY  
OF + CF +

2ad 6

licba pierwsza 1000 1000 - spakowana  
licba druga 0001 1000 - spakowana  
dodawane licb BCD.

1) Ropakowane druga w BX

MOV BX, 0x18 ..... |xxxx4444;  
ROL BX, 4 ..... xxxx |4444... ;  $0001 \text{ 1000} = 18$   
ROR BL, 4 ..... xxxx |.... 4444 ;  $1000 \text{ 1000} = 88$

0	...
1	...
2	..1
3	.11
4	.11
5	.11
6	.11
7	.11
8	1...
9	1..1
A	1..1
B	1..1
C	1..1
D	1..1
E	1..1
F	1..1

Ropakowyjacy - czyli roztadaj 8bitow na 16bitow.

to samo licba pierwsza

MOV AX, 88 ..... .... |xxxx4444  
ROL AX, 4 ..... xxxx |4444....  
ROR BL, 4 ..... xxxx |.... 4444

$$\begin{array}{r} 0808 \\ + 0108 \\ \hline 0910 \end{array}$$

ustawione flagi AF

2) dodajemy

$$\begin{array}{r} \dots aaaa | \dots bbbb \\ + \dots cccc | \dots dddd \\ \hline \dots ffff \dots 1 eeee \end{array}$$

AF ④

jeśli nastepne przesunięcie na 4 bitów to  
musimy wynik "wyroównać" ponownie na 16bitów  
zgodny z kodem BCD.

3) jeśli AF to:

AAA - wyrożnione po dodawaniu BCD.

0910 → AAA → AH+1, AL+6 + maska na 16bitów  
→ 1006 → AA many przesunięcia

→ (1).06 wynik - dodajemy 0x30 do każdej liczy  
wynika by dostac kod ASCII i wykonać

2cd 6a)

$$\begin{array}{r} \text{DX} \quad \text{AX} \\ 0 \ 1 \ 0 \ 6 \\ + \ 3 \ 0 \ 3 \ 0 \ 3 \ 0 \\ \hline \end{array}$$

30 31 30 36 = ASCII "0106" mala!