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)

Dlaczego zapis 16 jest tak popularny? Faktycznie "koduje" on grupy 4 bitów, i dalej się czytać i ogarnąć wrokiem nawet przy liczbach 64 bitowych.

np 00010100 + 00101000 = ? tego nie sposób czytać!

14+28 - to już dało się czytać.

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

Ponadto, czytając stan procesora 8086 możemy zobaczyć stany 4 rejestrów nawet na monitorze o niedużej ekranie 40 znaków np.

AX:	BX:	CX:	DX:
0000	FF04	2043	4076.

taki zapis jest zwięzły, a poza tym jest „elitarny" nie każdy go rozumie.
Wśród młodzieży modne jest h4ck mØw4 prawdopodobnie miała swoje korzenie właśnie w takim zapisie ØA04 + 0720 jest wyględa elegancko :). Podobnymi skrótami posługują się przy zapisie adresów IP4 i IP6.
Kolor w CSS można zapisać #6C6C6C albo rgba (196, 162, 148, 1);

---

algorytm konwersji 3 cyfrowego hex na ósemkowy oct. (ja we wtedy oct.) ABC – Hex defg – oct.

ABC = xxxx.xxxx.xxxx (2) =

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

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,
```

2ad 3) 2apis U1

$58_{(10)} = 3A_{(16)} = 0011.1010_{(b)}$

16 bit = 0x003Ah 32 bit x0000.003Ah

$-58_{(10)} = -3A_{(16)} = 1100.0101_{(b)}$

16 Bit = 0xFFC5h

32 Bit U_1 = 0xFFFFFFC5h

$-128_{(10)}$ = $128_{(10)}$ = x0100h

U_1 −128 = xFEFFh

16 Bit 0xFEFFh

32 Bit 0xFFFFFEFFh

$-1023_{(10)} \rightarrow 1023_{(10)}$ = 03FFh ↓ xFC00h

U_1 16 Bit = 0xFC00h

U_1 32 = 0xFFFFFC00h

Zad 2) Zapis w 16 bit i 32 bit U2

$-58_{(10)}$ =

16 bit = 0xFFC5h +1 = 0xFFC6h

32 bit = 0xFFFFFFC6h

$58_{(10)}$ = 3A U_1

16 bit = x003Ah U_2

32 bit x0000003Ah U_2

$-128_{(10)}$ U_1 16 = 0xFEFFh

16 bit +1 = 0xFF00h

32 bit = 0xFFFFFF00h

$-1023_{(10)}$ = 0xFC00h U_1

16 bit $U_2 = U_1$ + 1 = 0xFC01h

32 bit U_2 = 0x0000FC01h
