lab_2 – Wprowadzenie

Problem: prezentacja liczb jednobajtowych w postaci znakowej

liczba 123	=>	‘123’	dziesiętnie
	=>	‘7B’	szesnastkowo (hexadecymalnie)
	=>	‘0173’	ósemkowo (oktalnie)
	=>	‘01111011’	dwójkowo (binarnie)

Wybór reprezentacji:

- dziesiętna:
  najpopularniejsza, skomplikowany algorytm (wielokrotne dzielenie przez 10 i branie reszty z dzielenia), kolejne cyfry dziesiętnego rozwinięcia w odwrotnej kolejności (od tyłu), zmienna długość (1-3 cyfr)
- binarna:
  prosty algorytm (jeden bit = jedna cyfra), mała czytelność, największa długość, konieczność wykorzystania pętli
- ósemkowa:
  rzadko wykorzystywana, nierówny podział bajtu (pierwsza cyfra = 2 bity, druga i trzecia po 3 bity)
- szesnastkowa:
  popularna, najmniejsza długość (2 cyfry), każda połówka bajtu to jedna cyfra, „dziwne” cyfry (‘A’..’F’)

Zapis bajtu w postaci szesnastkowej:

- podział bajtu na połówki
- każda połówka daje jedną cyfrę szesnastkową (znak ‘0’..’9’,’A’..’F’)
- utworzenie napisu (ciągu znaków) poprzez zapisanie uzyskanych cyfr w odpowiedniej kolejności

Podział bajtu na połówki:

xxxxyyyy

AND 0x0F => 0000yyyy liczba [0..15]

AND 0xF0 => xxxx0000 liczba [0..240]

SHR n,val

SHIFT RIGHT (logiczne przesunięcie bitowe `val` w prawo o n bitów)

=> 0000xxxx liczba [0..15]

Zamiana połówki bajtu na cyfrę szesnastkową (znak):

aaaa – liczba [0..15]

0 => ‘0’ kod ASCII dec 48, hex 0x30

1 => ‘1’ kod ASCII dec 49, hex 0x31

…

9 => ‘9’ kod ASCII dec 57, hex 0x39

10 => ‘A’ kod ASCII dec 65, hex 0x41

…

15 => ‘F’ kod ASCII dec 70, hex 0x46

```
if aaaa < 10 then
    code = aaaa + 48    ( 0x30, ‘0’ )
else
    code = aaaa + 55    ( 65 – 10, ‘A’ – 10 )
```