lab_3 – Wprowadzenie

Problem: prezentacja dowolnych liczb w postaci znakowej szesnastkowej

Punkt wyjścia: funkcja zamieniająca bajt na postać szesnastkową (lab_2)

Wersja oryginalna (ver. 1):

```
      .type byte2hex,@function
byte2hex:
      MOVB  %al, tmp

      MOVB  tmp,%al           # first nibble
      ANDB  $0x0F,%al
      CMPB  $10,%al
      JB    digit1
      ADDB  $('A'-10),%al
      JMP   insert1
digit1:
      ADDB  $'0',%al
insert1:
      MOVB  %al,%ah

      MOVB  tmp,%al           # second nibble
      SHR   $4,%al
      CMPB  $10,%al
      JB    digit2
      ADDB  $('A'-10),%al
      JMP   insert2
digit2:
      ADDB  $'0',%al
insert2:
      RET
```

Kod: **17** instrukcji, dane: **0/1** bajt (zmienna **tmp**).

Wersja alternatywna (ver. 2):

```
      .type byte2hex,@function

byte2hex:
      MOVB  %al, tmp

      MOVB  tmp, %al              # first nibble
      ANDB  $0x0F, %al
      MOVZX %al, %rbx             # rbx = al; zeros in empty space
      MOVB  hex_digit(%rbx), %ah  # ah = hex_digit[ rbx ]

      MOVB  tmp, %al              # second nibble
```

```
      SHR   $4, %al
      MOVZX %al, %rbx               # rbx = al; zeros in empty space
      MOVB  hex_digit(%rbx), %al    # al = hex_digit[ rbx ]
      RET
```

Kod: **10** instrukcji, dane: **16/17** bajtów – tablica **hex_digit** (+**tmp**).

Wersja alternatywna (ver. 3):

```
      .type byte2hex,@function

byte2hex:
      MOVZX %al, %rbx                    # rbx = al; zeros in empty space
      MOVW  hex_digits(,%rbx,2), %ax     # ax = hex_digits[ rbx ]
      RET
```

Kod: **3** instrukcje, dane: **512** bajtów – tablica **hex_digits**.

Dyrektywy związane z kompilacją warunkową:

```
      .ifdef FUNC_V1
      …
      .endif
```

Elementy języka (dane i/lub instrukcje) zostaną skompilowane tylko wtedy, gdy w trakcie kompilacji będzie zdefiniowany symbol **FUNC_V1**!

Zdefiniowanie symbolu w trakcie kompilacji:

opcja kompilatora `--defsym SYMBOL=VALUE`

Dane:

.byte								B0
.word							B1	B0
.long					B3	B2	B1	B0
.quad	B7	B6	B5	B4	B3	B2	B1	B0

Algorytm:

- różna wielkość danych
- kolejność konwersji od **B0** do **Bk** (**k**=**0**,**1**,**3**,**7**)
- zapis wyniku konwersji od prawej do lewej

```
.byte     0xB0
.word     0xB1B0
.long     0xB3B2B1B0
.quad     0xB7B6B5B4B3B2B1B0
```

Realizacja:

- pętla **for k = 0 to size - 1**
  - pobranie bajtu **B0**
  - konwersja bajtu na dwa znaki hex
  - zapis dwóch znaków do pamięci (adres zawarty w **%rdi**)
  - zmniejszenie **%rdi** o **2** (przejście o 2 pozycje w lewo)
  - przesunięcie **B1** na miejsce **B0**, **B2** na miejsce **B1**, itd.
  - zwiększenie **k** o **1**

- problem: jak zwrócić rezultat konwersji? – funkcja dostaje jako argument adres miejsca w którym ma zapisać znaki i zapisuje je w kolejności od prawej do lewej (funkcja dostaje adres miejsca przeznaczonego na dwa znaki odpowiadające wartości najmniej znaczącego bajtu liczby i sama zmniejsza ten adres przy zapisywaniu w postaci szesnastkowej kolejnych konwertowanych bajtów)
- pozycja początkowa do zapisu znaków:

big_hex_str

byte	0	x	B0H	B0L														
word	0	x	B1H	B1L	B0H	B0L												
long	0	x	B3H	B3L	B2H	B2L	B1H	B1L	B0H	B0L								
quad	0	x	B7H	B7L	B6H	B6L	B5H	B5L	B4H	B4L	B3H	B3L	B2H	B2L	B1H	B1L	B0H	B0L
adres	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11	+12	+13	+14	+15	+16	+17

byte	%rdi = $big_hex_str + 2
word	%rdi = $big_hex_str + 4
long	%rdi = $big_hex_str + 8
quad	%rdi = $big_hex_str + 16