Arhitektura računara

Osnovni tipovi podataka u asembleru

Neoznačeni i označeni brojevi

- Jedina vrsta podataka koju procesor poznaje je ceo broj
- Brojevi mogu biti neoznačeni i označeni
- Značenje zavisi od programa, odnosno interpretacije

Neoznačeni i označeni brojevi

- Primeri:

10000001_2	-127_{10}	129_{10}
01100001_2	97_{10}	“a”

- Koriste se indikatori: *carry* i *overflow*

Karakteristike x86 procesora

- Svaki bajt ima svoju adresu
- *Little endian* - višebajtni brojevi se smeštaju od najmanje značajnog dela ka najviše značajnom
- Najviše jedan operand može biti pristupanje memoriji
  - u tom slučaju, drugi mora biti ili registar ili konstanta

Definisanje numeričkih podataka

- Podaci se smeštaju u segment podataka
- Definišu se:
  - **labelom** (ime za lokaciju u memoriji),
  - **direktivom veličine podatka**, i
  - **inicijalnom vrednošću**

Definisanje numeričkih podataka

- Direktive: **.byte**, **.word**, **.long**, **.quad**, ...
- Primer:

```
.section .data                      # sadržaj u memoriji u bajtovima

    bajt:
        .byte 0xff, 100             # 0xff 0x64
    rec:
        .word 0xee56, 2, 50         # 0x56 0xee 0x02 0x00 0x32 0x00
    duga_rec:
        .long 0xabcd1234            # 0x34 0x12 0xcd 0xab
```

Primeri upotrebe promenljivih

```
.section .data                          # sadržaj u memoriji u bajtovima
    bajt:
        .byte 0xff, 100                 # 0xff 0x64
    rec:
        .word 0xee56, 2, 50             # 0x56 0xee 0x02 0x00 0x32 0x00
    duga_rec:
        .long 0xabcd1234                # 0x34 0x12 0xcd 0xab
```

```
movl duga_rec, %eax     # 0xabcd1234 -> eax
movw %ax, rec           # sadržaj ax -> rec
movl $bajt, %eax        # adresa labele bajt -> eax
movw bajt, %ax          # 0x64ff -> ax
movl rec+2, %eax        # 0x320002 -> eax
movb duga_rec, %al      # 0x34 -> al
movl bajt, %eax         # 0xee5664ff -> eax
```

Definisanje i upotreba konstanti

```
const1 = 5
const2 = const1+4                  # const2 ima vrednost 9
const3 = const2*2                  # const3 ima vrednost 18
movl $const1+const2, %eax          # 14 -> eax
movl $const3*4, %ebx               # 72 -> ebx
movl $const1*%eax, %ebx            # GREŠKA! (ne postoji u x86 arh.)
```

- Aritmetičke operacije se mogu koristiti **samo za konstante**

Promenljive i konstante

- Primer 1:

```
max_vrednost = 20
vrednost: .long max_vrednost

movl $max_vrednost, %eax
```

- Primer 2:

```
a: .long 10
b = 15
…
movl $a, %eax    # adresa promenljive a -> eax
movl a, %eax     # vrednost promenljive a -> eax
movl $b, %eax    # vrednost konstante b -> eax
movl b, %eax     # GREŠKA!
                   (pokušaj pristupa lokaciji sa adresom 15)
```

Inicijalna vrednost promenljive

- Primer:

```
a: .long
b: .long 5

…

    movl a, %eax
    movl $7, a
    movl b, %eax
```

Prikaz promenljivih u dibageru

- **32-bitne promenljive**
  - klik na promenljivu, pa na dugme *Display*
- **16-bitne i 8-bitne promenljive**
  - klik na promenljivu, pa na jedno od dugmadi *WordH, WordD, WordU*, odnosno *ByteH, ByteD, ByteU*

Višestruka preciznost

- Koristi se kada je potrebno koristiti numeričke vrednosti koje ne mogu da stanu u jedan registar
- Vrednost se deli na delove koji mogu stati u registar, i to od najmanje značajne ka najviše značajnoj

Dvostruka preciznost na 80386

- 64-bitni brojevi se mogu definisati iz delova ili korišćenjem direktive **.quad**
- Primer:

```
a1: .long 20, 10          # vrednost 42949672980₁₀ = A00000014₁₆
a2: .quad 0xA00000014     # vrednost 42949672980₁₀ = A00000014₁₆
```

Dvostruka preciznost u dibageru

- *Data/Memory* dijalog omogućava, između ostalog, i prikaz vrednosti u dvostrukoj preciznosti
- Prikaz 64-bitne promenljive kroz *Data/Memory* dijalog:

  Examine=1, hex/decimal/unsigned, giants(8), &a1

  Examine=1, hex/decimal/unsigned, giants(8), &a2

Sabiranje u dvostrukoj preciznosti

- Prvo se sabiraju niži delovi, a zatim viši
- Da li pri sabiranju, odnosno oduzimanju, nižih delova treba uračunati prenos?
  - **add/adc**, odnosno **sub/sbb**
- Da li pri sabiranju, odnosno oduzimanju, viših delova treba uračunati prenos?
  - **add/adc**, odnosno **sub/sbb**
- Koje greške mogu nastati?

Sabiranje u dvostrukoj preciznosti

- **Neoznačeni brojevi**

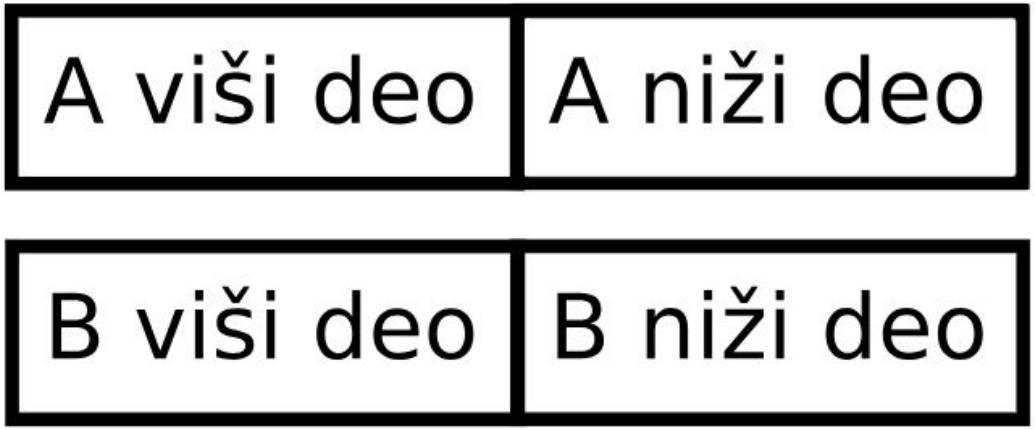
```
a: .quad 0x8000          # a = 0x8000 = 0x00008000
b: .long 0x8001, 1       # b = 0x100008001
greska: .byte 0
…
    movb $0, greska
    movl a, %eax
    addl b, %eax
    movl a+4, %edx
    adcl b+4, %edx
    jnc kraj
    incb greska
kraj:
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

Sabiranje u dvostrukoj preciznosti

- **Označeni brojevi**

```
a: .quad 0x8000          # a = 0x8000 = 0x00008000
b: .long 0x8001, 1       # b = 0x100008001
greska: .byte 0
…
    movb $0, greska
    movl a, %eax
    addl b, %eax
    movl a+4, %edx
    adcl b+4, %edx
    jno kraj
    incb greska
kraj:
    movl $1, %eax
    movl $0, %ebx
    int $0x80
```

Poređenje u dvostrukoj preciznosti




```
0x000189AB (0001 89AB)
0x00025678 (0002 5678)
```

Numerički tipovi podataka na C-u i asembleru

Veličina (biti)	Opseg	C	Asembler	Dibager
8	-128 do +127	char	.byte	bytes
8	0 do 255	unsigned char	.byte	bytes
16	-32768 do + 32767	short int	.word	halfwords
16	0 do 65535	unsigned short int	.word	halfwords
32	-2147483648 do +2147483647	int	.long	words
32	0 do 4294967295	unsigned int	.long	words
64	-2^{63} do $+2^{63}-1$	long long	.quad	giants
64	0 do $2^{64}-1$	unsigned long long	.quad	giants