

Assembler för INTEL/AMD 64 bitar (x64)

del 2

med lösningar



Fler kodexempel

- Fyra uppgifter
- Uppgift 3 och 4 viktiga inför Laboration 3!

Uppgift 1

Skriv en subrutin, `max(x,y)`, som bestämmer vilket av två heltal som är störst där

`x` = första heltalet

`y` = andra heltalet

returvärde = det största av de två talen `x` och `y`.

Rutinen ska anropas från följande C-program:

```
#include <stdio.h>
int max(int, int);

int main()
{
    int x,y,res;
    printf("Mata in två heltal\n");
    scanf("%d", &x);
    scanf("%d", &y);
    res = max(x,y);
    printf("Talet %d var störst av dem\n", res);
    return 0;
}
```

Noteringar:

`x` och `y` = 32-bit integer

`x` = `%edi`

`y` = `%esi`

`max` = `%eax`

OUT:	<u>rax</u>
PUSH/POP:	<u>rbx</u>
IN4:	<u>rcx</u>
IN3:	<u>rdx</u>
IN2:	<u>rsi</u>
IN1:	<u>rdi</u>
PUSH/POP:	<u>rbp</u>
(Stackpekare):	<u>rsp</u>
IN5:	r8
IN6:	r9
	r10
	r11
PUSH/POP:	r12
PUSH/POP:	r13
PUSH/POP:	r14
	r15

Scanf: `%d`=integer, `%f`=float, `%c`=char, `%s`=string.

& framför `x` och `y` betyder att man pekar på `x` resp. `y`

Uppgift 1, lösning

Noteringar:

x och y = 32-bit integer

x = %edi (jämför %rdi)

y = %esi (jämför %rsi)

max = %eax (jämför %rax)

```
.global max
```

```
max:
```

```
    cmpl    %esi, %edi    #jämför y och x
    cmovl   %esi, %edi    #flytta %esi (y) till %edi om större
                                #(annars ligger x kvar i %edi)
    movl    %edi, %eax    #uppdatera returvärdet %eax
    ret
```

cmov^l – Conditional move if less.

Uppgift 2

Antag att vi har en sekvens av n st positiva tal (array). Skriv en subrutin, `maxNum`, som ger oss det största talet i sekvensen.

Argument 1: Adress till talsekvensen

Argument 2: Antalet tal i sekvensen

Returvärde: Det största av talen i sekvensen

Rutinen ska ingå i ett bibliotek som ska anropas från följande C-progr.:

```
#include <stdio.h>
int maxNum(int *, int);
int main()
{
    int vect[5];
    int res;

    printf("Mata in fem heltal\n");
    scanf("%d", &vect[0]);
    scanf("%d", &vect[1]);
    scanf("%d", &vect[2]);
    scanf("%d", &vect[3]);
    scanf("%d", &vect[4]);
    res = maxNum(vect, 5);
    printf("Talet %d var störst av dem\n", res);
    return 0;
}
```

Noteringar:

`vect[5]` = 5 st 32-bit integer
`res` = 32-bit integer
`(%rdi)` pekar på `vect[0]`
`%esi` = 5
`maxNum` = `%eax`

OUT:	<code>rax</code>
PUSH/POP:	<code>rbx</code>
IN4:	<code>rcx</code>
IN3:	<code>rdx</code>
IN2:	<code>rsi</code>
IN1:	<code>rdi</code>
PUSH/POP:	<code>rbp</code>
(Stackpekare):	<code>rsp</code>
IN5:	<code>r8</code>
IN6:	<code>r9</code>
	<code>r10</code>
	<code>r11</code>
PUSH/POP:	<code>r12</code>
PUSH/POP:	<code>r13</code>
PUSH/POP:	<code>r14</code>
	<code>r15</code>

Uppgift 2, lösning

Noteringar:

vect[5] = 5 st 32-bit integer
res = 32-bit integer
(%rdi) pekar på vect[0]
%esi = 5
maxNum = %eax

```
.global maxNum
```

```
maxNum:
```

```
    movl (%rdi), %eax # Start: vect[0] läggs i %eax1 (nuv. tal)
```

```
lLoop:
```

```
    addq $4, %rdi      # Flytta pekaren (%rdi) till nästa element i vect[]  
                        # dvs -> vect[1] -> vect[2] -> vect[3] -> vect[4]
```

```
    cmpl %eax, (%rdi)  # Jämför nuv. tal med nästa element i vect[]
```

```
    jgl lLabel         # Om nuv. tal > nästa element, hoppa till lLabel
```

```
    movl (%rdi), %eax  # annars uppdatera returvärde med största hittills
```

```
lLabel: decl %esi      # minska %esi 5 -> 4 -> 3 -> 2 -> 1
```

```
    cmpl $1, %esi      #
```

```
    jgl lLoop          # Färdig? dvs %esi=1?
```

```
    ret
```

jgl label (jump to label där l = less than)

jg label (jump to label där g = greater than)

Uppgift 3

Vi har en teckensträng som startar med siffror avslutas med "icke-siffra".

Ex: "1234X", "122Y", "13323_" eller "123123C"

Skriv en rutin, `readInt`, som går igenom strängen och returnerar ett heltal (det tal vars representation utgörs av siffrorna i strängen).

Argument: Adress till strängen

Returvärde: Utläst heltal

Rutinen ska kunna anropas från följande C-program:

```
#include <stdio.h>
int readInt(char *);

int main()
{
    char str [10];
    int res;
    printf("Mata in ett tal! Avsluta med #!\n");
    fgets(str, 12, stdin);
    res = readInt(str);
    printf("Talet är: %d \n", res);
    return 0;
}
```

Noteringar:

str[10] = 10 st char
res = 32-bit integer
(%rdi) pekar på str[0]
readInt = %eax

'0' = 48 ₁₀	'A' = 65 ₁₀
⋮	⋮
'9' = 57 ₁₀	'Z' = 90 ₁₀

Uppgift 3

lös

Noteringar:

str[10] = 10 st char
res = 32-bit integer
(%rdi) pekar på str[0]
readInt = %eax

.global readInt

readInt:

movq \$0, %rax # Returvärde

lNumber:

cmpb \$'0', (%rdi)

j1 lEnd

cmpb \$'9', (%rdi)

jg lEnd

movzbq (%rdi), %r10

subq \$'0', %r10

imulq \$10, %rax

addq %r10, %rax

incq %rdi

jmp lNumber

lEnd:

ret

checka om str[?] har lägre ASCII-siffra än '0'
(vilket innebär "icke-siffra" -> avslutning)
checka om str[?] har högre ASCII-siffra än '9'
(vilket innebär "icke-siffra" -> avslutning)

Gör ASCII-kod till 64-bit med inledande nollor
Drag ifrån ASCII-kod för '0' -> finns siffra som tal

Multiplicerar med 10 på tidigare summa
och addera ny entalssiffra

flytta fram till nästa tecken, str[?]
och fortsätt leta efter siffra eller avslut

j1 label (jump to label där l = less than)

jg label (jump to label där g = greater than)

movzbq S, R (finns bara som 64-bit) gör R ← ZeroExtend(S),
dvs fyller ut med nollor i de högsta 64-8=56 bitarna till vänster
om ASCII-tecknet 'o' innan det hamnar i r8, 64-bit

Uppgift 4

Vi har en teckensträng som startar med siffror avslutas med "icke-siffra".

Den kan innehålla ett godtyckligt antal blanktecken (noll eller flera) framför talet samt att talet kan inledas med ett tecken '+' eller '-' före första siffran.

Ex: " +1234X", " -122Y", " 13323 " eller "123123C"

Modifiera uppgift 3 med den röda inringningen!

Att-göra-lista:

- Införa en teckenindikator (ifall vi upptäcker minustecken)
- Leta upp blanktecken och kasta dem
- Leta upp ev. '+'
- Leta upp ev. '-'
- Om minus: Teckenindikator = 1, annars 0
- Kör uppgift 3
- Ändra ev. tecken till negativt

Noteringar:

str[10] = 10 st char
res = 32-bit integer
(%rdi) pekar på str[0]
readInt = %eax



- Införa en teckenindikator
- Leta upp blanktecken och kasta
- Leta upp ev. '+'
- Leta upp ev. '-'
- Om minus: Teckenindikator = 1, annars 0
- Kör uppgift 3
- Ändra ev. tecken till negativt

Uppgift 4 lös

Noteringar:

str[10] = 10 st char
res = 32-bit integer
(%rdi) pekar på str[0]
readInt = %eax

```
.global readInt
readInt:
    movq $0, %rax
    movq $0, %r11

lBlancCheck:
    cmpb $' ', (%rdi)
    jne lSignPlus
    incq %rdi
    jmp lBlancCheck

lSignPlus:
    cmpb $'+', (%rdi)
    jne lSignMinus
    incq %rdi
    jmp lNumber

lSignMinus:
    cmpb $'-', (%rdi)
    jne lNumber
    movq $1, %r11
    incq %rdi

lNumber:
```

Gulmarkerat = oförändrat

Returvärde

Teckenindikator (0 = positivt tal)

str[?]=blank?

om inte blank -> checka om tecken (dvs '+', '-', siffra)

om blank -> flytta fram till nästa tecken, str[?]

str[?]='+'?

om inte '+' -> checka om '-'

om '+' -> flytta fram till nästa tecken, str[?]

och leta efter siffror

str[?]='-'?

om inte '-' -> checka om siffror

Håll reda på att talet är negativt (endast om '-')

flytta fram till nästa tecken, str[?] och leta siffror

Kom ihåg till nästa sida: %rax=0 (heltalet) samt %r11=1 om "-" annars %r11=0



- Införa en teckenindikator
- Leta upp blanktecken och kasta
- Leta upp ev. '+'
- Leta upp ev. '-'
- Om minus: Teckenindikator = 1, annars 0
- Kör uppgift 3
- Ändra ev. tecken till negativt

Uppgift 4

lös

Noteringar:

str[10] = 10 st char
res = 32-bit integer
(%rdi) pekar på str[0]
readInt = %eax

(forts. från föreg. sida)

Gulmarkerat = oförändrat (frånsett lEnd -> lNAN = Not_A_Number)

lNumber:

cmpb \$'0', (%rdi)

j1 lNAN

cmpb \$'9', (%rdi)

jg lNAN

movzbq (%rdi), %r10

subq \$'0', %r10

imulq \$10, %rax

addq %r10, %rax

incq %rdi

jmp lNumber

lNAN:

cmpq \$1, %r11

jne lEnd

negq %rax

lEnd:

ret

checka om str[?] har lägre ASCII-siffra än '0'

(vilket innebär "icke-siffra" -> avslutning)

checka om str[?] har högre ASCII-siffra än '9'

(vilket innebär "icke-siffra" -> avslutning)

Gör ASCII-kod till 64-bit med inledande nollor

Drag ifrån ASCII-kod för '0' -> finns siffra som tal

Multiplicerar med 10 på tidigare summa

och addera ny entalssiffra

flytta fram till nästa tecken, str[?]

och fortsätt leta efter siffra eller avslut

Minustecken?

om inte minustecken -> klar

om minustecken -> byt tecken så det blir minus

movzbq S, R (finns bara som 64-bit) gör R ← ZeroExtend(S),
dvs fyller ut med nollor i de högsta 64-8=56 bitarna till vänster
om ASCII-tecknet 'o' innan det hamnar i r8, 64-bit