Asemblersko programiranje za x86_64 arhitekturu
2. deo

Najvažniji kodovi (flegovi) uslova

- **Jednobitni registri (ali okupljeni u registar RFLAGS)**
  - CF Carry Flag (za neoznačene) SF Sign Flag (za označene brojeve)
  - ZF Zero Flag OF Overflow Flag (za označene brojeve)

- **Postavljaju se (kao uzgredni efekt) aritmetičkih operacija**

  Primer: add Dest,Src ↔ t = a+b

  CF na 1 ukoliko postoji prenos/pozajmica sa najznačajnijeg bita t (detektuje prekoračenje opsega neoznačenih brojeva)

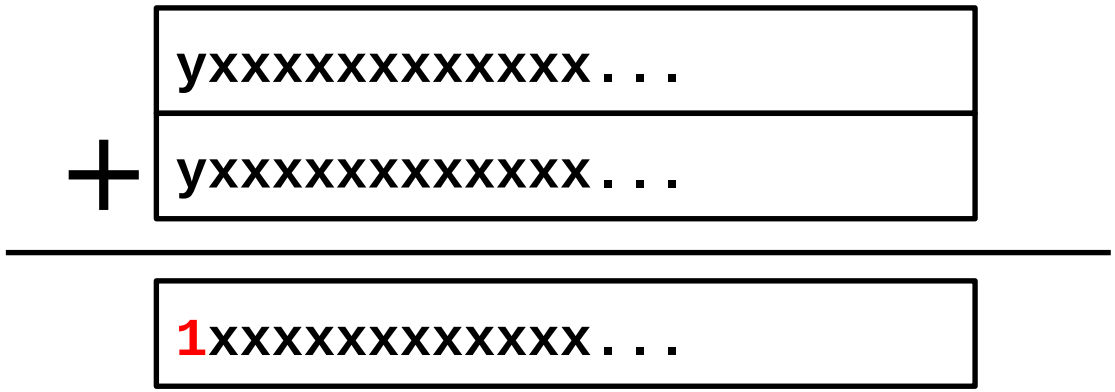
  ZF na 1 ako je t == 0

  SF na 1 ako je t < 0 (pri čemu se t tretira kao označen, u komplementu 2)

ZF na 1 kada

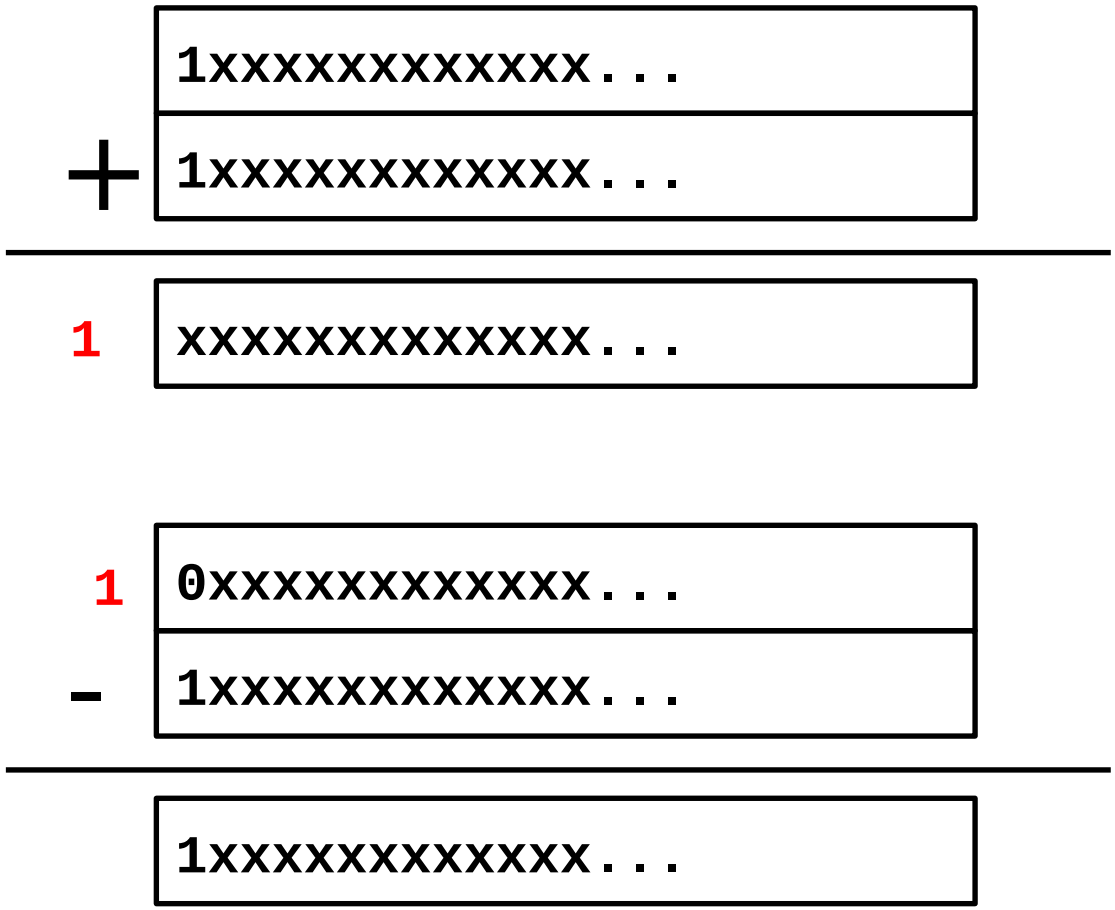
000000000000…00000000000

SF na 1 kada



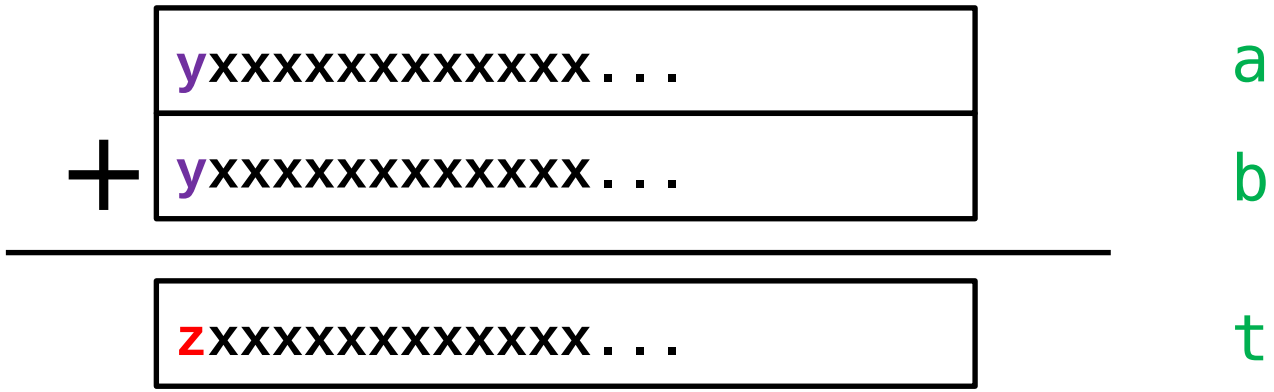
Za označenu aritmetiku, ovo znači da je rezultat negativan broj

CF na 1 kada



Za neoznačenu aritmetmetiku, CF na 1 znači prekoračenje opsega

OF na 1 kada



$$z = \sim y$$

(a>0 && b>0 && t<0) || (a<0 && b<0 && t>=0)

Za neoznačenu aritmetmetiku, OF na 1 znači prekoračenje opsega

Kodovi (flegovi) uslova

- **Ažuriranje flegova od strane instrukcija poređenja**
  - cmp Src1, Src2
  - cmp a,b računa se a-b ali se rezultat nigde ne pamti

  - Međutim, flegovi se ažuriraju:
  - CF na 1 ukoliko postoji prenos/pozajmica sa najznačajnijeg bita t (kod neoznačenog poređenja znači da je a < b)
  - ZF na 1 ako je a == b
  - SF na 1 ako je (a-b) < 0 (za označene brojeve)
  - OF na 1 u slučaju prekoračenja kod označenih brojeva
    (a>0 && b<0 && (a-b)<0) || (a<0 && b>0 && (a-b)>0)

Kodovi (flegovi) uslova

- **Ažuriranje flegova instrukcijom test**
  - test Src1, Src2
    - test a,b sračunava a & b bez pamćenja rezultata
  - Postavlja kodove uslova na osnovu vrednosti Src1 & Src2
  - Često je jedan od operanada bit maska da proveri vrednost određenog bita kod drugog operanda
  - ZF na 1 kada a & b == 0
  - SF na 1 kada a&b < 0

Često u programu provera da li je vrednost registra jednaka

```
test  rax, rax
```

Kodovi (flegovi) uslova

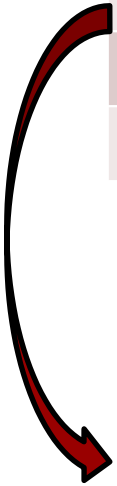
- **Set instrukcije**
  - `setX` r/m8: Postavlja bajt najmanje težine odredišta r ili m na 0 ili 1
    zavisno od vrednosti kombinacije kodova uslova
  - Ne menja preostalih 7 bajtova odredišta

SetX	Condition	Description
sete	ZF	Equal / Zero
setne	~ZF	Not Equal / Not Zero
sets	SF	Negative
setns	~SF	Nonnegative
setg	~(SF^OF)&~ZF	Greater (signed)
setge	~(SF^OF)	Greater or Equal (signed)
setl	SF^OF	Less (signed)
setle	(SF^OF)|ZF	Less or Equal (signed)
seta	~CF&~ZF	Above (unsigned)
setb	CF	Below (unsigned)

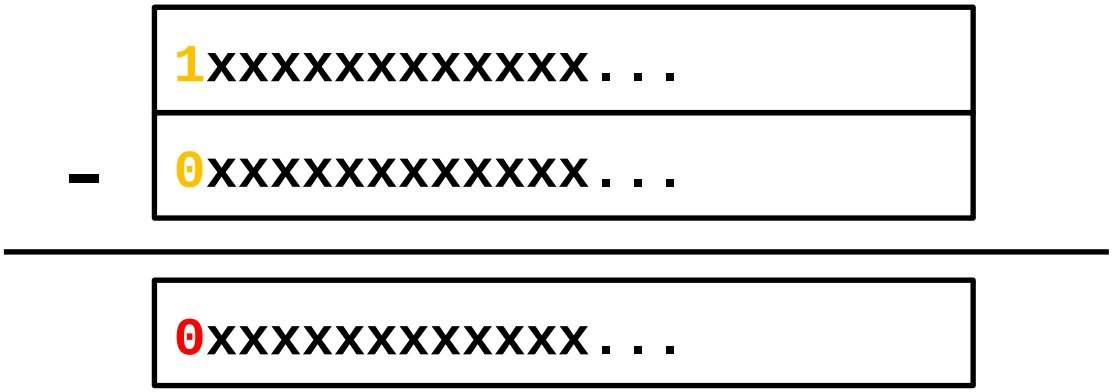
Primer: setl (< za označene brojeve)

- **Uslov: SF^OF**

SF	OF	SF ^ OF	Objašnjenje
0	0	0	Nema prekoračenja, a SF ukazuje da nije <
1	0	1	Nema prekoračenja, a SF ukazuje da je <
0	1	1	Prekoračenje, a SF kaže sa negativne strane, jeste <
1	1	0	Prekoračenje, a SF ukazuje sa + strane, tako da nije <



negative overflow case



Primer upotrebe setX instrukcija

- **setX instrukcije:**
  - Postavljaju jedan bajt na 0 ili 1 u zavisnosti od vrednosti kombinovanih kodova uslova
- **Operand r treba da je jedan od bajtovskih registara**
  - Ne menja preostale bajtove većeg registra koji sadrži r
  - Ako želimo da se ceo veliki registar postavi upotrebićemo movzx
    - e takođe

```
int gt (long x, long y)
{
  return x > y;
}
```

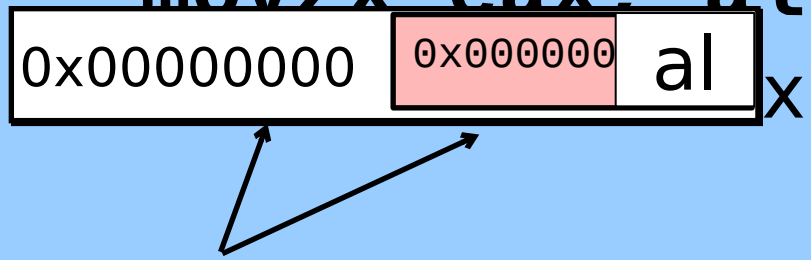
Registar	Upotreba
rdi	Argument x
rsi	Argument y
rax	Return value

```
cmp   %rdi, %rsi   # Compare x:y
   setg   al          # Set when >
   movzx eax, al      # Zero rest of rax
   ret
```

Primer upotrebe setX instrukcija

Zapamtiti “čudnovato” ponašanje **movzx**

movzx eax, al



Postavlja sve 0 i u gornjoj polovini relevantnog 64bitnog registra

Programski skokovi

- **jX instrukcije**
  - Skok na odredište u kodu u zavisnosti od vrednosti (ranije postavljenih uslovnih kodova

jX	Condition	Description
jmp	1	Unconditional
je	ZF	Equal / Zero
jne	~ZF	Not Equal / Not Zero
js	SF	Negative
jns	~SF	Nonnegative
jg	~(SF^OF)&~ZF	Greater (signed)
jge	~(SF^OF)	Greater or Equal (signed)
jl	SF^OF	Less (signed)
jle	(SF^OF)|ZF	Less or Equal (signed)
ja	~CF&~ZF	Above (unsigned)
jb	CF	Below (unsigned)

Primer uslovnog grananja

- Prevođenje komandom

```
gcc -masm=intel -Og -S  primer.c
```

```
long absdiff
  (long x, long y)
{
  long result;
  if (x > y)
    result = x-y;
  else
    result = y-x;
  return result;
}
```

```
absdiff:
    cmp     rdi, rsi  # x:y
    jg      .L4
    mov     rax, rsi
    sub     rax, rdi
    ret
.L4:                  # x > y
    mov     rax, rdi
    sub     rax, rsi
    ret
```

Register	Use(s)
rdi	Argument x
rsi	Argument y
rax	Return value

Uslovne mov instrukcije

- **cmovX dest, src instrukcije**
  - Izvršavanje instrukcije:

    if (X) Dest Src
  - Podržane od 1995 u x86 procesorima
  - GCC kompajler ih takođe koristi pri optimizaciji
- **Korist od cmov**
  - Skokovi ometaju tok instrukcija kroz procesorski pajplajn
  - Uslovni mov ne zahteva transfer kontrole toka, izvršava se u sekvenci sa drugim

C kod

```
val = Test
    ? Then_Expr
    : Else_Expr;
```

Goto Verzija C

```
result = Then_Expr;
eval = Else_Expr;
nt = !Test;
if (nt) result = eval;
return result;
```

Primer absdiff sa uslovnim move

- **Prevođenje komandom**

Ranije bilo -Og

```
gcc -masm=intel –O2 -S  primer.c
```

```
long absdiff
  (long x, long y)
{
    long result;
    if (x > y)
        result = x-y;
    else
        result = y-x;
    return result;
}
```

Registar	Upotreba
rdi	Argument x
rsi	Argument y
rax	Return value

```
absdiff:
    mov   rdx, rdi # x
    mov   rax, rsi # y
    sub   rdx, rsi # result = x-y
    sub   rax, rdi # eval = y-x
    cmp   rdi, rsi # compare x and y
    cmovg rax, rdx # if >, result=eval
    ret
```

Prevođenje kontrolnih konstrukcija Ca
“Do-While” prevođenje

C kod

```
do
  Body
  while (Test);
```

C kod sa If i

```
loop:
  Body
  if (Test)
    goto loop
```

- **Body:{**

```
        Statement1;
        Statement2;
          …
        Statementn;
      }
```

Prevođenje kontrolnih konstrukcija
“While” prevođenje varijanta #1

- “Skok-u-sredinu” varijanta prevođenja
- Kada se zada **-Og**

While verzija

```
while (Test)
  Body
```

IF-Goto verzija

```
  goto test;
loop:
  Body
test:
  if (Test)
    goto loop;
done:
```

Prevođenje kontrolnih konstrukcija
“While” prevođenje varijanta #2

While verzija

```
while (Test)
  Body
```

- “Do-while” konverzija
- Kada se zada **–01**

Do-While Verzija

```
  if (!Test)
    goto done;
  do
    Body
    while(Test);
done:
```

Goto Verzija

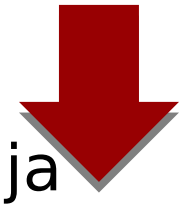
```
  if (!Test)
    goto done;
loop:
  Body
  if (Test)
    goto loop;
done:
```

Prevođenje kontrolnih konstrukcija

“For” petlja While petlja

For verzija

```
for (Init; Test; Update )
        Body
```



While Verzija

```
Init;
while (Test ) {
    Body
    Update;
}
```

Primer sa “Do-While” petljom (-O1)

- Count number of 1’s in argument x (“popcount”)

C kod

```
long pcount_do
  (unsigned long x) {
  long result = 0;
  do {
    result += x & 0x1;
    x >>= 1;
  } while (x);
  return result;
}
```

IF-Goto

```
long pcount_goto
  (unsigned long x) {
  long result = 0;
 loop:
  result += x & 0x1;
  x >>= 1;
  if(x) goto loop;
  return result;
}
```

```
   mov     eax, 0        #   result = 0
.L2:                      # loop:
   mov     rdx, rdi
   and     edx, 1        #   t = x & 0x1
   add     rax, rdx    #   result += t
   shr     rdi           #   x >>= 1
   jne      .L2          #   if(x) goto loop
   rep ret
```

Register	Use(s)
rdi	Argument x
rax	result