# Bitovske operacije

Računarstvo i informatika, III godina

Februar 2013. godine

## 1 Operacije pomeranja

Asemblerski jezik omogućava programeru da manipuliše individualnim bitovima podataka. Uobičajena operacija je pomeranje (*shift*). Operacija pomeranja pomera pozicije bitova nekog podatka. Pomeranje se može izvoditi ulevo ili udesno.

### 1.1 Logičko pomeranje

Logičko pomeranje je najjedostavnija vrsta pomeranja. Primer pomeranja za jedno mesto je dat na Slici 1.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Original | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| Left shifted | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| Right shifted | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |

Slika 1: Logičko pomeranje

Treba primetiti da je vrednost novih, dolazećih bitova uvek jednaka nuli. U asembleru za Intel procesore se za logičko pomeranje koriste instrukcije `SHL` i `SHR`. Ove instrukcije omogućavaju pomeranje za bilo koji broj pozicija. Broj poziija može biti konstanta ili može biti smešten u `CL` registar. Poslednji bit podatka se čuva u *carry flag*-u. Evo nekoliko primera:

```
mov ax, 0C123h
shl ax, 1          ; pomeri jedan bit ulevo,  ax=8246h, CF=1
shr ax, 1          ; pomeri jedan bit udesno, ax=4123h, CF=0
shr ax, 1          ; pomeri jedan bit udesno, ax=2091h, CF=1
mov ax, 0C123h
shl ax, 2          ; pomeri dva bita ulevo, ax=448Ch, CF=1
mov cl, 3
shr ax, cl         ; pomeri 3 bita udesno, ax=0091h, CF=1
```

### 1.2 Upotreba pomeranja

Jedna od uobičajenih upotreba pomeranja je brzo izvođenje operacija množenja i deljenja stepenom dvojke. Recimo, u dekadnom sistemu, množenje i deljenje sa stepenima broja 10 se može obaviti prostim pomeranjem cifara. Slično važi i za sistem sa osnovom 2 i množenje i deljenje stepenima dvojke. Na primer, ukoliko hoćemo da dupliramo binarni broj $1011_2$ (11 dekadno), treba ga pomeriti za jedno mesto ulevo da bi se dobilo $10110_2$ (22 dekadno). Ukoliko se želi deljenje sa 2, pomera se jedno mesto udesno, ako se deli sa 4 pomera se dva mesta udesno, sa 8 ide 3 mesta udesno itd. Ovaj način je mnogo jednostavniji i brži od glomaznih `MUL` i `DIV` instrukcija.

Treba zapaziti da logička pomeranja možemo vršiti samo na neoznačenim brojevima. Ako npr. uzmemo 2-bajtnu vrednost `FFFF` (-1 označeno) i pomerimo jedno mesto udesno, dobijamo `7FFFF`, što je jednako vrednosti +32767. Za označene vrednosti se koristi drugi tip operacije pomeranja.

## 1.3 Aritmetičko pomeranje

Aritmetičko pomeranje je vrsta pomeranja koja se koristi za brzo množenje i deljenje označenih brojeva stepenima dvojke.

**SAL** (*Shift Arithmetic Left*) - je instrukcija koja je sinonim za `SHL`. Asemblira se u potpuno isti mašinski kod kao instrukcija `SHL`. Rezultat će biti tačan sve dok se bit za znak ne menja.

**SAR** (*Shift Arithmetic Right*) - je nova instrukcija koja ne pomera bit znaka operanda. Ostali bitovi se normalno pomeraju, osim što su bitovi koji nadolaze sa leva kopije bita znaka. Dakle, ako se pomera jedan bajt, u stvari se pomera donjih 7 bita. Kao i kod ostalih instukcija, poslednji pomeren bit se čuva u *carry flag*-u.

```
mov ax, 0C123h
sal ax, 1              ; ax=8246h, CF=1
sal ax, 1              ; ax=048Ch, CF=1
sar ax, 2              ; ax=0123h, CF=0
```

## 1.4 Rotacije

Rotacije rade potpuno isto kao pomeranja, s tim što se bitovi koji se gube na jednom kraju, sada pojavljuju na drugom kraju podatka. Dakle, podatak se tretira kao ciklična struktura. Dve najjednostavnije instrukcije za rotaciju su `ROL` i `ROR` koje služe za rotaciju ulevo i udesno, respektivno. *Carry flag* takođe čuva poslednji pomeren bit.

```
mov ax, 0C123h
rol ax, 1             ; ax=8247h, CF=1
rol ax, 1             ; ax=048Fh, CF=1
rol ax, 1             ; ax=091Eh, CF=0
ror ax, 2             ; ax=8247h, CF=1
ror ax, 1             ; ax=C123h, CF=1
```

Takođe, tu su i dve dodatne instrukcije za rotaciju, `RCL` i `RCR`, koje kao deo podatka uzimaju i *carry flag*. Na primer, kada se registar AX rotira pomoću ovih instrukcija, rotira se podatak od 17 bita (`AX` + `CF`):

```
mov ax, 0C123h
clc                   ; obriši vrednost CF, CF=0
rcl ax, 1             ; ax=8246h, CF=1
rcl ax, 1             ; ax=048Dh, CF=1
rcl ax, 1             ; ax=091Bh, CF=0
rcr ax, 2             ; ax=8246h, CF=1
rcr ax, 1             ; ax=C123h, CF=0
```

Primer 1.1 (Prebrojavanje bitova) *Napisati asemblerski program koji prebrojava koliko jedinica ima binarni zapis broja koji se nalazi u `EAX` registru.*

```
    mov bl, 0
    mov ecx, 32
petlja:
    shl eax, 1
    jnc preskok
    inc bl
preskok:
    loop petlja
```

# 2 Bitovske logičke operacije

Prilikom upotrebe bilo koje od bitovskih operacija koje slede, bitno je naglasiti da se one izvršavaju na svakom paru bitova nezavisno i paralelno.

## 2.1 AND operacija

```
mov ax, 0C123h
and ax, 82F6h           ; ax = 8022h
```

## 2.2 OR operacija

```
mov ax, 0C123h
or ax, 0E831h           ; ax = E933h
```

## 2.3 XOR operacija

```
mov ax, 0C123h
xor ax, 0E831h           ; ax = 2912h
```

## 2.4 NOT operacija

NOT operacija je unarna, što zanči da uzima samo jedan operand. NOT operacija je, u stvari nepotpuni komplement, za rauliku od NEG operacije koja predstavlja potpuni komplement.

```
mov ax, 0C123h
not ax                  ; ax = 3EDCh
```

## 2.5 TEST instrukcija

TEST instrukcija se ponaša potpuno isto kao AND, ali nigde ne pohranjuje rezultat. Ona naime, poput CMP instrukcije, samo postavi odgovarajuće vrednosti pojedinačnih zastavica unutar FLAGS registra. Na primer, ako je rezultat nula, postavlja se ZF na jedinicu.

## 2.6 Jednostavna upotreba binarnih operacija

Evo nekoliko primera upotrebe binarnih operacija u realnom radu:

1. **Uključivanje $i$-tog bita** - izvršiti OR sa $2^i$ (što je, u stvari, binarni broj sa samo $i$-tim bitom 1, a sve ostalo su nule)
2. **Isključivanje $i$-tog bita** - izvršiti AND sa binarnim brojem u kome je samo $i$-ti bit 0, a sve ostalo jedinice
3. **Komplement $i$-tog bita** - izvršiti XOR sa $2^i$ (što je, u stvari, binarni broj sa samo $i$-tim bitom 1, a sve ostalo su nule)

# 3 Zadaci

1. Štampati koliko jedinica ima binarni zapis svih označenih brojeva od -16 do 15. Prebrojavanje jedinica obaviti u proceduri, a štampu u glavnom programu.

2. Učitati prirodni broj sa tastature i smestiti ga u AL registar. Odrediti koliko je 1-bitnih pomeranja ulevo potrebno da bi se promenila cifra u CF registru. Na primer, za broj 00001010 (10 dekadno), potrebno je 5 pomeranja, a za broj 11101100 potrebno je 4 pomeranja.

3. Napisati program koji za učitani 32-bitni prirodni broj ispisuje koliko ima jedinica na parnim mestima u njegovoj binarnoj reprezentaciji. Za prebrojavanje jedinca koristiti proceduru.

# Literatura

[1] Paul Carter, PC Assembly Language, 2006.