Ponašanje sklopovlja procesora pri izvođenju svake instrukcije

```
ponavljati {
    dohvatiti iz spremnika instrukciju na koju pokazuje programsko brojilo;
    dekodirati instrukciju, odrediti operaciju koju treba izvesti;
    povećati sadržaj programskog brojila tako da pokazuje na sljedeću instrukciju;
    odrediti odakle dolaze operandi i gdje se pohranjuje rezultat;
    operande dovesti na aritmetičko-logičku jedinku, izvesti zadanu operaciju;
    pohraniti rezultat u odredište;
}
dok je (procesor uključen);
```

Procesor pri izvođenju instrukcije **bezuvjetnog skoka**

```
ponavljati {
    dohvatiti iz spremnika instrukciju na koju pokazuje programsko brojilo;
    dekodirati instrukciju, odrediti operaciju koju treba izvesti;
    povećati sadržaj programskog brojila tako da pokazuje na sljedeću instrukciju;
    ako je (dekodirana instrukcija skoka)
        iz pomaknuća zapisanog u instrukciji odrediti adresu i smjestiti tu
        adresu u programsko brojilo;
    inače
        obaviti instrukciju na način određen dekodiranim operacijskim kodom;
}
dok je (procesor uključen);
```

Procesor pri izvođenju instrukcije **uvjetovanog skoka**

```
ponavljati {
    dohvatiti iz spremnika instrukciju na koju pokazuje programsko brojilo;
    dekodirati instrukciju, odrediti operaciju koju treba izvesti;
    povećati sadržaj programskog brojila tako da pokazuje na sljedeću instrukciju;
    ako je (dekodirana instrukcija uvjetovanog skoka ∧ uvjet ispunjen)
        iz pomaknuća zapisanog u instrukciji odrediti adresu i smjestiti
        tu adresu u programsko brojilo;
    inače
        obaviti instrukciju na način određen dekodiranim operacijskim kodom;
}
dok je (procesor uključen);
```

Procesor u izvođenju **poziva potprograma**

```
ponavljati {
    dohvatiti iz spremnika instrukciju na koju poka uje programsko brojilo;
    dekodirati instrukciju, odrediti operaciju koju treba izvesti;
    povećati sadržaj programskog brojila tako da pokazuje na sljedeću instrukciju;
    ako je (dekodirana instrukcija poziva potprograma) {
        pohraniti sadržaj programskog brojila na stog;
        smanjiti sadržaj registra SP tako da pokazuje na sljedeće pra no mjesto;
        iz adresnog dijela instrukcije odrediti adresu početka potprograma;
        smjestiti tu adresu u programsko brojilo;
    }
    inače
        obaviti instrukciju na način određen dekodiranim operacijskim kodom;
}
dok je (procesor uključen);
```

Procesor u izvođenju **povratka iz potprograma**

```
ponavljati {
    dohvatiti iz spremnika instrukciju na koju pokazuje programsko brojilo;
    povećati sadržaj programskog brojila tako da pokazuje na sljedeću instrukciju;
    dekodirati instrukciju, odrediti operaciju koju treba izvesti;
    ako je (dekodirana instrukcija povratka iz potprograma) {
        povećati sadržaj registra kazaljke stoga SP;
        premjestiti sadržaj sa vrha stoga na koji pokazuje SP u programsko
        brojilo;
    }
    inače
        obaviti instrukciju na način određen dekodiranim operacijskim kodom;
}
dok je (procesor uključen);
```

Prihvat jednog znaka od ulazne naprave

1) strojni kod

```
        ADR r1,RS    ;
        ADR r2,PR    ;
PETLJA  LDR r0,[r1]  ; čitati RS
        CMP r0,#0    ; (ostali bitovi su 0)
        BEQ PETLJA   ;
        LDR r0,[r2]  ; čitati PR
```

2) pesudokod

```
pročitati registar RS;
dok je (ZASTAVICA == 0){
    pročitati registar stanja RS;
}
pročitati registar PR; //time se automatski briše ZASTAVICA
```

Mehanizam međusobnog isključivanja

```
LDR  r0,[r1]      ; prenijeti u r0 sadržaj s adrese koja se nalazi u r1
STR  r0,[r1]      ; prenijeti sadržaj r0 na adresu koja se nalazi u r1
ADR  r1,SIMB_IME  ; smjestiti u r1 adresu čije je simboličko ime SIMB_IME
CMP  r0,#0        ; usporedba r0 s nulom
BEQ  PETLJA       ; skočiti na PETLJA ako prethodna usporedba daje jednakost
BAL  PETLJA       ; skočiti uvijek
```

Prekidni način rada procesora

```
ponavljati {
    dohvatiti iz spremnika instrukciju na koju pokazuje programsko brojilo;
    dekodirati instrukciju, odrediti operaciju koju treba izvesti;
    povećati sadržaj programskog brojila tako da pokazuje na sljedeću instrukciju;
    odrediti odakle dolaze operandi i kuda se pohranjuje rezultat;
    operande dovesti na aritmetičko-logičku jedinku, izvesti zadanu operaciju;
    pohraniti rezultat u odredište;
    ako je (prekidni signal postavljen) {
        onemogućiti daljnje prekidanje;
        prebaciti adresiranje u sustavski adresni prostor i aktivirati sustavsku
        kazaljku stoga;
        pohraniti programsko brojilo na sustavski stog;
        staviti u programsko brojilo adresu potprograma za obradu prekida;
    }
}
dok je (procesor uključen);
```

Ispitni lanac: ako imamo više pristpnih sklopova koji onda mogu generirati više prekida

```
(pojavio se prekidni signal, onemogućeno je prekidanje i programsko brojilo se
nalazi na sustavskom stogu)
{
    pohraniti kontekst na sustavski stog;
    pročitati registar stanja pristupa RS_N;
    ako je (ZASTAVICA_N = 1)
        skočiti na obradu prekida_N;
    pročitati registar stanja pristupa RS_N-1;
```

```
    ako je (ZASTAVICA_N-1 = 1)
        skočiti na obradu prekida_N-1;
        :
    pročitati registar stanja pristupa RS_1;
    ako je (ZASTAVICA_1 = 1)
        skočiti na obradu prekida_1;
    obnoviti kontekst sa sustavskog stoga;
    omogućiti prekidanje;
    vratiti se iz prekidnog načina;
}
```

Odsječak programa za obradu prekida pojedinog pristupnog sklopa

```
(obrada prekida pristupa I)
{
    pročitati podatkovni registar PR_I
    (ili upisati podatak u podatkovni registar PR_I);
    poništiti zastavicu u registru stanja RS_I;
    obaviti sve dodatne aktivnosti;
          :
    obnoviti kontekst sa sustavskog stoga;
    omogućiti prekidanje;
    vratiti se iz prekidnog načina;
}
```

Proces pri pojavi prekida na svom ulazu na kraju izvođenja tekuće instrukcije obavlja slijedeće:

```
ako je (prekidni signal postavljen) {
    zabraniti daljnje prekidanje;
    prebaciti adresiranje u sustavski adresni prostor i aktivirati sustavsku
    kazaljku stoga;
    pohraniti programsko brojilo i sve ostale registre na sustavski stog;
    postaviti signal PRIHVAT;
    sa sabirnice preuzeti sadržaj i iz njega odrediti adresu odsječka za obradu
    prekida;
    staviti tu adresu u programsko brojilo;
}
```

- „**vratiti se u prekinutu dretvu**“

```
obnoviti kontekst sa sustavskog stoga;
omogućiti prekidanje;
vratiti se u način rada prekinute dretve i obnoviti programsko brojilo sa sustavskog
stoga;
```

- **obrada prekida sa indeksom I**

```
dohvatiti registar T_P i staviti ga na sustavski stog;
zapisati u T_P[I] = 1;
omogućiti prekidanje;
obaviti obradu prekida I;
onemogućiti prekidanje;
obnoviti sa sustavskog stoga registar T_P;
vratiti se u prekinutu dretvu;
```

Unutar pristupnog sklopa ua neposredni pristup spremniku obavlja se sljedeći dio koda:
(DMA- Direct memory access)

```
dok je (BR > 0) {
    zatražiti sabirnicu;
    čekati na dodjelu sabirnice;
    postaviti na adresni dio sabirnice sadržaj registra AR;
    prenijeti na tu adresu sadržaj podatkovnog PR (ili obrnuto);
    AR = AR + 1;
    BR = BR - 1;
}
postaviti signal PREKID;
```

MEĐUSOBNO ISKLJUČIVANJE

1) PRVI POKUŠAJ (NE VALJA)
   - **varijabla ZASTAVICA** (ako je 0 onda je ulaz u KO slobodan, ako je 1 onda nije)

```
dok je (1) {
    čitati varijablu ZASTAVICA;
    dok je (ZASTAVICA != 0) {
        čitati varijablu ZASTAVICA;
    }
    ZASTAVICA = 1;
    kritični odsječak;
    ZASTAVICA = 0;
    nekritični osječak;
}
```

```
        ADR  r1,ZASTAVICA  ;
PETLJA  LDR  r0,[r1]       ;  čitati ZASTAVICA
        CMP  r0,#1         ;
        BEQ  PETLJA        ;
        STR  #1,[r1]       ;  ZASTAVICA = 1
        prva instrukcija kritičnog odsječka;
        .
        .
        .
        zadnja instrukcija kritičnog odsječka;
        STR  #0,[r1]       ;  ZASTAVICA = 0
        prva instrukcija nekritičnog odsječka;
        .
        .
        .
        zadnja instrukcija nekritičnog odsječka;
        BAL  PETLJA        ;
```

- ne valja za slučaj da obe dretve u isto vrijeme žele ući u KO, obe će pročitat da je zastavica 0 i da je ulaz slobodan

2) DRUGI POKUŠAJ (NE VALJA)
   - **varijabla PRAVO**: poprima vrijednost indeksa dretvi (I), ako je PRAVO==I onda u KO može ući dretva sa tim indeksom
   - nije ispunjen uvjet da dretva koja zastane u svom NKO ne smije sprijčti drugu dretvu da uđe u svoj KO

```
dok je (1) {
    čitati varijablu PRAVO;
    dok je (PRAVO != I) {
        čitati varijablu PRAVO;
    }
    kritični odsječak;
    PRAVO = J;
    nekritični osječak;
}
```

3) TREĆI POKUŠAJ (NE VALJA)
   - dvi zastavice (**ZASTAVICA[I] i ZASTAVICA[J]** koja služi za čitanje stanaj suparničke dretve)
   - to znači da će jedna dretva moći ući u KO samo onda ako se druga dretva nalazi u svom NKO
   - isti problem kao kod prvog pokušaja

```
dok je (1) {
    čitati varijablu ZASTAVICA[J];
    dok je (ZASTAVICA[J] != 0) {
        čitati varijablu ZASTAVICA[J];
    }
    ZASTAVICA[I] = 1;
    kritični odsječak;
    ZASTAVICA[I] = 0;
    nekritični odsječak;
}
```

4) ČETVRTI POKUŠAJ (NI ON NE VALJA ☹)
   - mogli bi se pokušat ispravit prošli pokušaj tako da dretva prije ispitivanja suparničke zastavice podigne zastavicu i tako naznači da ona želi u KO, ali zasto bi bilo jednostavno kad može i komplicirano tako da ni ovo ne valja
   - u ovom primjeru može doći do toga da obje dretve uđu u ptelju čekalicu i njoj ostanu beskonačno (**Dead lock – mrtvo zaglavljenje** lol)

```
dok je (1) {
    ZASTAVICA[I] = 1;
    čitati varijablu ZASTAVICA[J];
    dok je (ZASTAVICA[J] != 0) {
        čitati varijablu ZASTAVICA[J];
    }
    kritični odsječak;
    ZASTAVICA[I] = 0;
    nekritični osjećak;
}
```

5) PETI POKUŠAJ (NE VALJA)
   - moglo bi se napravit da kad dretva koja želi ući u KO podigne svoju zastavicu i potvrdi da je suprotna dretva to isto napravila, da se onda privremeno spusti zastavica prve dretve i da ona čeka dok druga dretva to isto ne napravi
   - ovo ne valja jer se opet dretve mogu odvijati istodobno u više situacija:
     - prvi je da obe dretve u dva uzastopna sabirnička ciklusa podignu zastavice
     - drugi je da obe dretve u dva uzaspona sabirnička ciklusa pročitaju zastaice suprotnih dretvi
     - obe dretve ustanove da je suprostavljena zastavica podignuta i u dva uzastopna sabirnička ciklusa spuste svoje zastavice
   - ovim načinom obe bi ostale u petlji čekalici beskonačno

```
dok je (1) {
    ZASTAVICA[I] = 1;
    čitati varijablu ZASTAVICA[J];
    dok je (ZASTAVICA[J] != 0) {
        ZASTAVICA[I] = 0;
        čitati varijablu ZASTAVICA[J];
        dok je (ZASTAVICA[J] != 0) {
            čitati varijablu ZASTAVICA[J];
        }
        ZASTAVICA[I] = 1;
        čitati varijablu ZASTAVICA[J];
    }
    kritični odsječak;
    ZASTAVICA[I] = 0;
    nekritični osječak;
}
```

6) ŠESTI POKUŠAJ – **DEKKEROV PUSTUPAK** (OVAJ NAPOKON VALJA)
   - kombinacija drugog i petog pokušaja
   - varijablu PRAVO ispitujemo onda i samo onda kada obje dretve konkuriraju za ulazak u kritični odsječak, tada ta varijabla može odrediti redosljed ulaska i pomoći u rješenju problema međusobnog isključivanja

```
dok je (1) {
    ZASTAVICA[I] = 1;
    čitati varijablu ZASTAVICA[J];
    dok je (ZASTAVICA[J] != 0) {
        čitati varijablu PRAVO;
        ako je (PRAVO != I) {
            ZASTAVICA[I] = 0;
            dok je (PRAVO != I){
                čitati varijablu PRAVO;
            }
            ZASTAVICA[I] = 1;
        }
        čitati varijablu ZASTAVICA[J]
    }
    kritični odsječak;
    PRAVO = J;
    ZASTAVICA[I] = 0;
    nekritični osječak;
}
```

- varijabla PRAVO se ipituje onda i samo onda kada obe dretve istovremeno podignu svoje zastavice
- dretve se mogu izvoditi samo naizmjence, izbjegava se problem koji smo imali u drgom pokušaju

PETTERSONOV POSTUPAK – POJEDNOSTAVLJENJE DEKKEROVA POSTUPKA

- varijablu PRAVO preimenujemo u **NE_PRAVO**
- svaka dretva koja želi ući u KO postavlja svoju zastavicu i nakon toga u NE_PRAVO zapisuje svoj indeks
- ako obje dretve isstovremeno žele ući u KO, onda će varijabla NE_PRAVO poprimiti vrijednost one dretve čiji je procesor zadnji dobio pravo pristupa na sabirnicu
- vreijednost koja je prije toga bila stavljena u varijablu je izgubljena
- nakon toga dretva čeka u petlji tako dugo dok suprostavljena dretva ne spusti svoju zastavicu
- imati na umu da ovo vrijedi za međusobno isključivanje 2 dretve

```
dok je (1) {
    ZASTAVICA[I] = 1;
    NE_PRAVO = I;
    čitati varijablu ZASTAVICA[J];
    čitati varijablu NE_PRAVO;
    dok je ((NE_PRAVO = = I) && (ZASTAVICA[J] = = 1)) {
        čitati varijablu ZASTAVICA[J];
        čitati varijablu NE_PRAVO;
    }
    kritični odsječak;
    ZASTAVICA[I] = 0;
    nekritični osječak;
}
```

LAMPORTOV PROTOKOL – MEĐUSOBNO ISKLJUČIVANJE VIŠE DRETVI

- pekarski algoritam
- u zajedničkom spreminku se nalazi
  - BROJ[N]
  - ULAZ[N]
- globalni spremnik sadrži jednu varijablu ZADANI_BROJ koja je na početku inicijalizirana 0

```
dok je (1) {
    ULAZ[I] = 1;
    čitati ZADNJI_BROJ;
    BROJ[I] = ZADNJI_BROJ + 1;        } ulaz kroz vrata
    ZADNJI_BROJ = BROJ[I];
    ULAZ[I] = 0;
    za (J = 0, J < N, J++) {                                   } predavanje svih dretvi
        čitati varijablu ULAZ[J];
        dok je (ULAZ[J] = = 1) {       } pričekati ako dretva J upravo ulazi kroz vrata
            čitati varijablu ULAZ[J]
        }
        čitati varijablu BROJ[J];
        dok je ((BROJ[J] != 0) && ((BROJ[J],J) < (BROJ[I],I)) {
            čitati varijablu BROJ[J]
        }
    }
    kritični odsječak;
    BROJ[I] = 0;
    nekritični osječak;
}
```

- kada dretva želi u KO do svog broja dolazi na sljedeci način

```
ULAZ[I] = 1;
čitati ZADNJI_BROJ;
BROJ[I] = ZADNJI_BROJ + 1;
ZADNJI_BROJ = BROJ[I];
ULAZ[I] = 0;
```

NEDJELJIVE INSTRUKCIJE:

1) **TAS (Test And Set)**
   a. u prvom ciklusu dohvate sadržaj adresina lokacije i smjesta ga u jedan od registara, u drugom ciklusu pohranjuju u tu lokaciju vrijednost 1

```
dok je (1) {
        TAS ZASTAVICA;
        dok je (ZASTAVICA !=0) {
                TAS ZASTAVICA;
        }
        kritični odsječak;
        ZASTAVICA = 0;
        nekritični odječak;
}
```

2) **SWAP (zamijeni)**
   a. u prvom ciklusu dobave sardžaj adresirane lokacije i smjeste ga u jedan od registara, dok u drugom ciklusu pohranjuju u tu lokaciju vrijednost koja je prije toga bila pohranjena u tom registru

3) **FETCH-AND-ADD**
   a. u prvom ciklusu pročitaju sadržaj adresirane lokacije i sprema ga u registar, a u drugom ciklusu pohranjuju na tu lokacije tu vrijednost uvećanu za 1

FUNKCIJE ZA BINARNI SEMAFOR

1) **ispitati_BSEM(i)**
   - dretva poziva kada želi ući u KO

```
j-funkcija Ispitati_Bsem(I) {
    pohraniti kontekst u opisnik Aktivna_D;
    ako je (Bsem[I].v == 1) {
        Bsem[I].v = 0;
        obnoviti kontekst iz opisnika Aktivna_D;
        omogućiti prekidanje;
        vratiti se iz prekidnog načina;
    }
    inače {
        premjestiti opisnik iz reda Aktivna_D u red Bsem[I];
        aktivirati prvu dretvu iz reda Pripravne_D;
    }
}
```

2) **postaviti_BSEM(i)**
   - poziva kada dretva izlazi iz KO

```
j-funkcija Postaviti_Bsem(I) {
    pohraniti kontekst u opisnik Aktivna_D;
    premjestiti opisnik iz reda Aktivna_D u red Pripravne_D;
    ako je (red Bsem[I] neprazan) {
        premjestiti prvi opisnik iz reda Bsem[I] u red Pripravne_D;
    }
    inače {
        Bsem[I].v = 1;
    }
    aktivirati prvu dretvu iz reda Pripravne_D;
}
```

FUNCKIJE ZA OPĆI SEMAFOR

(u suvremenim operacijskim sutavima, vrijednost ne može biti negativna)

1) ispitati_OSEM(j)

```
j-funkcija Ispitati_Osem(J) {
    pohraniti kontekst u opisnik Aktivna_D;
    ako je (Osem[J].v >= 1) {
        Osem[J].v = Osem[J].v - 1;
        obnoviti kontekst iz opisnika Aktivna_D;
        omogućiti prekidanje;
        v atiti se iz prekidnog načina;
    }
    inače {
        premjestiti opisnik iz reda Aktivna_D u red Osem[J];
        aktivirati prvu dretvu iz reda Pripravne_D;
    }
}
```

2) postaviti_OSEM(j)

```
j-funkcija Postaviti_Osem(J) {
    pohraniti kontekst u opisnik Aktivna_D;
    premjestiti opisnik iz reda Aktivna_D u red Pripravne_D;
    ako je (red Osem[J] neprazan) {
        premjestiti prvi opisnik iz reda Osem[J] u red Pripravne_D;
    }
    inače {
        Osem[J].v = Osem[J].v + 1;
    }
    aktivirati prvu dretvu iz reda Pripravne_D;
}
```

FUNCKIJE ZA OPĆI SEMAFOR (mehanizam koji je opisao Dijskre)

- ovo nije klasični opći semafor OSEM!
- može poprimiti negativne vrijednosti
- nazivamo ga OS

1) ispitati_OS(j)

```
j-funkcija Ispitati_Os(J) {
    pohraniti kontekst u opisnik Aktivna_D;
    Os[J].v = Os[J].v - 1;
    ako je (Os[J].v >= 0) {
        obnoviti kontekst iz opisnika Aktivna_D;
        omogućiti prekidanje;
        vratiti se iz prekidnog načina;
    }
    inače {
        premjestiti opisnik iz reda Aktivna_D u red Os[J];
        aktivirati prvu dretvu iz reda Pripravne_D;
    }
}
```

2) postaviti_OS(j)

```
j-funkcija Postaviti_Os(J) {
    pohraniti kontekst u opisnik Aktivna_D;
    premjestiti opisnik iz reda Aktivna_D u red Pripravne_D;
    Os[J].v = Os[J].v + 1;
    ako je (Os[J].v == 0) ∧ (red Os[J] neprazan)) {
        premjestiti opisnik iz reda Os[J] u red Pripravne_D;
    }
    aktivirati prvu dretvu iz reda Pripravne_D;
}
```

FUNCKIJE ZA OSTVARIVANJE KAŠNJENA

1) dretva može zatražiti da sebe zakasni za M perioda otkucaja sata, poziva se funkcija zakasniti_sebe(M)

```
j-funkcija Zakasniti_sebe(M) {
      pohraniti kontekst u opisnik Aktivna_D;
      uvrstiti opisnik iz reda Aktivna_D u red Odgođene_D;
      aktivirati prvu dretvu iz reda Pripravne_D;
}
```

2) aktivna dretva može tražiti i da neka druga dretva s indeksom L zakasni. to se ostvaruje funkcijom zakasniti(L, M)

```
j-funkcija Zakasniti(L,M) {
      pohraniti kontekst u opisnik Aktivna_D;
      premjestiti opisnik iz reda Aktivna_D u red Pripravne_D;
      pronaći opisnik dretve L u listi Postojeće_D;
      ako je (dretva L nije pasivna) {
            dojaviti pogrešku;
      }
      inače {
            uvrstiti opisnik dretve L u red Odgođene_D;
      }
      aktivirati prvu dretvu iz reda Pripravne_D;
}
```

3) deblokiranje dretvi koje se nalaze u redu odgođene_d obvavit ce funkcija prekidom od sata (otkucaj_sata)

```
j-funkcija Otkucaj_sata {
      pohraniti kontekst u opisnik Aktivna_D;
      premjestiti opisnik iz reda Aktivna_D u red Pripravne_D;
      ako je (red Odgođene_D neprazan) {
            umanjiti u prvom opisniku reda Odgođene_D sadržaj lokacije
            Zadano_kašnjenje za jedan;
            ako je (rezultat smanjenja jednak nuli) {
                  premjestiti prvi opisnik iz reda Odgođene_D u red Pripravne_D
                  (a i sve sljedeće opisnike ako u lokaciji
                  Zadano_kašnjenje imaju zapisanu vrijednost nula);
            }
      }
      aktivirati prvu dretvu iz reda Pripravne_D;
}
```

FUNKCIJE ZA OBAVLJANJE ULAZNO- IZLAZNIH OPERACIJA

1) funkcija kojom započinje neka ulazno izlazna operacija rezultirat ce funkcijom zapocni_UI

```
j-funkcija Započeti_UI(K) {
    pohraniti kontekst u opisnik Aktivna_D;
    premjestiti opisnik iz reda Aktivna_D u red UI[K];
    pokrenuti ulazno-izlaznu operaciju na napravi K;
    aktivirati prvu dretvu iz reda Pripravne_D;
}
```

2) prekid od ulazno izlazne naprave pokrenit će jezgrinu funckiju prekid_UI

```
j-funkcija Prekid_UI(K) {
    pohraniti kontekst u opisnik Aktivna_D;
    premjestiti opisnik iz reda Aktivna_D u red Pripravne_D;
    premjestiti opisnik iz reda UI[K] u red Pripravne_D;
    aktivirati prvu dretvu iz reda Pripravne_D;
}
```

JEZGRINE FUNKCIJE ZA OSTVARANJE MONITORA

1) Ući_u_monitor(M)
   - po djelovanju je jednaka funckiji za ispitivanje binarnog semafora
   - jedina razlika je u tome što je Monitor(M) povezan sa redovima uvjeta

```
j-funkcija Ući_u_monitor(M) {
    pohraniti kontekst u opisnik Aktivna_D;
    ako je (Monitor[M].v == 1) {
        Monitor[M].v = 0;
        obnoviti kontekst iz opisnika Aktivna_D;
        omogućiti prekidanje;
        vratiti se iz prekidnog načina;
    }
    inače {
        premjestiti opisnik iz reda Aktivna_D u red Monitor[M];
        aktivirati prvu dretvu iz reda Pripravne_D;
    }
}
```

2) Izaći_iz_monitora(M)

```
j-funkcija Izaći_iz_monitora(M) {
    pohraniti kontekst u opisnik Aktivna_D;
    premjestiti opisnik iz reda Aktivna_D u red Pripravne_D;
    ako je (red Monitor[M] neprazan) {
        premjestiti prvi opisnik iz reda Monitor[M] u red Pripravne_D;
    }
    inače {
        Monitor[M].v = 1;
    }
    aktivirati prvu dretvu iz reda Pripravne_D;
}
```

3) uvrsti_u_red_uvjeta(M,K)
   - jezgrina funkcija za blokiranje dretvi unutar monitora pozivat će se unutar neke monitorske funkcije samo onda kada dretvu treba blokirati
   - ispitivanje uvjeta M se obavlja u korisničkom načinu rada
   - uvjet može biti i složen, a ne samo jednostavan kao što je slučaj kod binarnog semafora

```
j-funkcija Uvrstiti_u_red_uvjeta(M,K) {
    pohraniti kontekst u opisnik Aktivna_D;
    premjestiti opisnik iz reda Aktivna_D u Red_uvjeta[M,K];
    ako je (red Monitor[M] neprazan) {
        premjestiti prvi opisnik iz reda Monior[M] u red Pripravne_D;
    }
    inače {
        Monitor[M].v = 1;
    }
    aktivirati prvu dretvu iz reda Pripravne_D;
}
```

4) osloboditi_iz_reda_uvjeta(M,K)
   - jezgrina funckija za deblokiranje dretve koja je bila uvrštena u neki od redova uvjeta
   - dretva koja se deblokira i ona koja je tražila deblokiranje napuštaju monitor
   - monitorske funckije moraju bit napisane tako da se deblokiranje obavlja na kraju
   - svaka monitorksa funckija mora završiti ili pozivom jezgrine funckije za izlazak iz monitora, ili pozivom funckije za deblokiranje neke dretve ili funkcijom za blokiranje dretve

```
j-funkcija Osloboditi_iz_reda_uvjeta(M,K) {
    pohraniti kontekst u opisnik Aktivna_D;
    premjestiti opisnik iz reda Aktivna_D u red Pripravne_D;
    ako je (Red_uvjeta[M,K] neprazan)) {
        premjestiti prvi opisnik iz reda Red_uvjeta[M,K] u red Pripravne_D;
    }
    ako je (red Monitor[M] neprazan) {
        premjestiti prvi opisnik iz reda Monior[M] u red Pripravne_D;
    }
    inače {
        Monitor[M].v = 1;
    }
    aktivirati prvu dretvu iz reda Pripravne_D;
}
```