Operacijski sustavi - 1. Kolokvij

1. Model jednostavnog računala

1.1. Von Neumannov model računala

Von Neumannov model utvrđuje da svako računalo mora imati sljedeće dijelove:

- *Ulazni dio* preko kojeg se iz okoline u spremnik unose podaci i instrukcije programa
- *Izlazni dio* preko kojeg u okolinu prenose rezultati programa
- *Radni ili glavni spremnik* u koji se pohranjuju svi podaci i instrukcije programa uneseni izvana, kao i rezultati djelovanja instrukcija
- *Aritmetičko logičku jedinicu* koja može izvoditi instrukcijama zadane aritmetičke i logičke operacije
- *Upravljačku jedinicu* koja dohvaća instrukcije iz spremnika, dekodira ih i na temelju toga upravlja aritmetičko logičkom jedinicom, te ulaznim i izlaznim dijelovima

Središnji dio računala je *spremnik*. U njega dolaze svi podaci i instrukcije preko ulaznog dijelam, te rezultati operacija koje izvodi aritmetičko logička jedinica. Izlazna jedinica šalje dobivene rezultate iz spremnika u okolinu. Upravljačka jedinica iz spremnika dohvaća instrukcije, koje dekodira i zatim određuje operacije koje mora izvesti aritmetičko logička jedinica. Danas se upravljčka jedinica i aritmetičko logička jedinica spajaju u jednu cjelinu, koja se zove procesor. Upravljačka jedinica šalje upravljačke signale svim ostalim djelovima u računalu.

1.2. Sabirnica

Zajednički snop vodića na koji su spojeni svi dijelovi računala. Sabirnica osim vodića ima i sabirnički sklop koji pomaže kod ostvarivanja veza. U svakom trenutku samo jedna komponenta može dijeliti podatke pomoću sabirnice. Pristup sabirnici se zbog toga izvodi naizmjenice. Takva podjela vremena na sabirnici naziva se sabirnički ciklus. U jednom sabirničkom ciklusu ostvaruje se jedna veza, odnosno prijenos jednog sadržaja.

Ako svaki sabirnički ciklus traje 30 nanosekundi i svakim prijenosom prenose se 4 bajta (32-bitna arhitektura), tada se u jednoj sekundi može napraviti 33,000,000 prijenosa, odnosno 133 MB/s.

Vrste sabirnica: (Pazi, Von Neumannov model rečunala ima samu jednu sabirnicu, ne sve 3)

- *Adresna sabirnica* - Koristi se za prijenos adresa iz procesora prema memoriji ili ulazno-izlaznim uređajima. Procesor na adresnu sabirnicu stavi adresu dijela memorije kojem želi pristupiti. Širina adresne sabirnice ovisi o arhitekturi procesora (na primjer u 32-bitnoj arhitekturi može se adresirati 2^32 memorijskih lokacija)

- *Podatkovna sabirnica* - Koristi se za prijenos podataka između procesora, memorije i ulazno-izlaznih uređaja. Širina podatkovne sabirnice određuje koliko se podataka može prenijeti u jednom sabirničkom ciklusu.
- *Upravljačka sabirnica* - Prenosi upravljačke signale koji koordiniraju rad svih dijelova računala.

1.3. Procesor

Osnovna svojstva i ponašanje procesora određeni su skupom registara i skupom instrukcija koje procesor može obaviti. Registri služe za pohranjivanje svih informacijskih sadržaja koji ulaze i izlaze iz procesora i u njemu se transformiraju. Skup instrukcija određen je izvedbom aritmetičko logičke i upravljačke jedinice procesora.

Registri i njihova uloga:

- *Adresni međuregistar* služi za adresiranje spremnika i ostalih dijelova računala.
- *Podatkovni međuregistar* je posrednik za razmjenu sadržaja između procesora i ostalih dijelova računala. Svi podaci koji se žele prenijeti iz procesora na sabirnicu prolaze kroz podatkovni međuregistar.
- U *instrukcijski registar* prenosi se instrukcija dobavljena iz spremnika. Bitovi instrukcije dovode se na upravljačku jedinicu koja iz njih ustanovi koju instrukciju procesor treba obaviti.
- *Registar kazaljke stoga (engl. Stack pointer)* služi za poseban način adresiranja spremnika. U spremniku se rezervira posebna skupina uzastopnih bajtova i zapiše u registar kazaljke stoga najviša adresa te skupine. Pomoću tog registra ostvaruje se pohranjivanje podataka na stog.
- *Programsko brojilo (engl. Program counter)* je registar koji sadržava adresu sljedeće instrukcije koja se treba obaviti. Njega upravljačka jedinica automatski inkrementira jer se instrukcije pohranjene u spremniku izvode jedna iza druge.
- Bitovi *registra stanja* služe za zapisivanje različitih zastavica koje označavaju ispravnost ili neispravnost rezultata i operacija, kao i način rada procesora (korisnički, sustavski…). Vrijednosti tih zastavica su uvjeti na temelju kojih upravljački sklop određuje daljnji tijek odvijanja programa.
- *Skup registara opće namjene* služi za pohranjivanje operanada i rezulatata, te za pripremanje adresa budućih pristupa do spremnika.

Upravljačka jedinica dekodira instrukciju koja se nalazi u instrukcijskom registru, povezuje unutarnje procesorske sabirnice i daje ALU informaciju o vrsti operacije koja se mora odraditi.

Svi dijelovi računala rade u ritmu koji određuje generator takta. Sve pojave unutar procesora sinkronizirane su s generatorom takta. Generator takta određuje brzinu rada procesora.

Aritmetičko logička jedinica uzima ulaze iz unitarnjih sabirnica S1 i S2 i pohran-

juje rezultat aritmetičke ili logičke operacije na unutarnju sabirnicu S3. Upravljačka jedinica, osim što određuje operaciju, određuje i odakle dolaze operandi i kamo se sprema rezultat.

Procesor se može promatrati kao automat koji nakon pokretanja trajno izvodi instrukciju za instrukcijom.

Pseudokod:

```
ponavljati {
  dohvatiti instrukciju; // Na koju pokazuje PC
  dekodirati instrukciju;
  PC++; // Povecati PC tako da pokazuje na sljedecu instrukciju

  operande dovesti na ALU;
  izvesti operaciju;
  pohraniti rezultat;
} dok je (procesor uključen);
```

Izvođenje instrukcije dijeli se na 3 faze:

1. Dohvat instrukcije (engl. fetch)
   - Sadržaj programskog brojila postavi se u adresni međuregistar
   - Upravljačka jedinica pokreće aktivnost dohvata instrukcije iz spremnika
   - Instrukcija dolazi u podatkovni međuregistar i iz njega u instrukcijski registar
2. Dekodiranje instrukcije (engl. decode)
   - Utvrđuje na temelju dijela bitova instukcije (op kod) operaciju koju treba provesti
   - Povećava sadržaj programskog brojila tako da pokazuje na sljedeću instrukciju
   - Šalje upravljačke signale ALU kako bi ona znala koju operaciju treba provesti
   - Na temelju dijela bitova instrukcije (adresni dio instrukcije) ustanovljuje iz kojih registara dolaze operandi i gdje treba pohraniti rezultat
   - Ako adresni dio instrukcije određuje da operand dolazi iz spremnika, onda se adresa operanda prebacuje u adresni međuregistar i pokreće dohvatanje operanda iz spremnika
3. Obavljanje instrukcije (engl. execute)
   - ALU obavlja zadanu operaciju i pohranjuje razultat preko sabirnice S3 u odredište
   - Vrijednosti pojedinih zastavica koje ovise o dobivenom rezultatu pohranjuju se u registar stanja

Osim vrsta registara koje pojedini procesor ima, procesori se također mogu razlikovati i po broju instrukcija koje mogu izvoditi (sjeti se IRS CISC, RISC). Također, pokazalo se da je korisno imati instrukcije za skokove i povratak iz

potprograma, jer se neki programski zadaci često ponavljaju. (ovo pitaj AI samo ak nije jasno kak skokovi rade)

1.4. Program, proces i dretva

Program je niz instrukcija zapisan u nekom obliku. To može biti na papiru, datoteci itd. On je samo tvorevina. Tek kad se smjesti u radni spremnik i pokrene onda dobiva nova, dinamička obilježja i postaje proces. Proces je radno okruženje u kojem se izvodi program.Obuhvaća ne samo izvođenje instrukcija, nego i strukture podataka, strukture za pristup datotekama, ima vlastiti stog itd. Operacijski sustav mora osigurati sve uvjete za odvijanje procesa. Pokretanjem programa mora se pokrenuti **najmanje** jedan proces. Niz instrukcija koje su trenutno o izvođenju naziva se dretva. Svaki proces mora imati najmanje jednu dretvu. Više detalja kasnije.

Razlike u korištenju procesa i dretvi:

- Procesi:
  - Koristi se kad posao mora biti izoliran od ostalih poslova u računalu
  - Procesi mogu međusobno komunicirati pomoću mehanizama operacijskog sustava, ali to je složenije nego kod dretvi
- Dretve:
  - Koristi se kad se posao može paralelno odraditi i kad poslovi moraju međusobno komunicirati
  - Dretve lako komuniciraju preko zajedničke memorije unutar procesa

2. Ulazno-izlazni podsustav

Po načinu rada, ulazno-izlazne naprave i njihovi pristupni sklopovi mogu se podijeliti na dvije skupine:

- Naprave kod kojih se prenose *pojedinačni znakovi (bajtovi)*
- Naprave kod kojih se prenose *blokovi znakova (niz uzastopnih bajtova)*

2.1. Radno čekanje

[!INFO] Pročitaj knjigu

TLDR

```
ponavljati {
  citati registar RS;
  dok je (ZASTAVICA == 0) {
    citati regustar RS;
  }
  citati znak iz podatkovnog registra PR;
  pohraniti procitani znak u spremnik;
} dok je (procesor ukljucen);
```

Program se sastoji od 2 dijela: dijela koji se sastoje od petlje čekalice u kojoj procesor samo troši vrijeme i dijela u kojem se znak prenosi iz podatkovnog resistra pristupnog sklopa u jedan bajt spremnika. Koristan posao obavlja samo drugi dio odsječka, a sastoji se od svega desetak strojnih instrukcija. To znači da samo 0.1% vremena procesor radi koristan posao, a ostatak od 99.9% utrošen je na čekanje. (Jako shit)

2.2. Prekidni način rada

Umjesto da procesor čeka primitak novog znaka u podatkovni registar, on može normalno raditi, a kad ulazna naprava napiše novi znak u podatkovni registar, tada postavi zastavicu i generira električni signal. Električni signal dolazi do procesora posebnim vodićem. Sad procesor umjesto da čeka, jednostavno na kraju svake instrukcije provjeri postoji li signal na tom vodiću.

Pojava prekidnog signala prebacuje procesor u sustavski način rada. U sustavskom načinu rada mogu se pozivati potprogrami koji čine jezgru operacijskog sustava, pa se zbog toga taj način rada procesora još naziva i jezgreni način rada. Korisnički programi obavljaju se u takozvanom koristničkom načinu rada. Način rada može se pročitati iz registra stanja, to jest određen je nekim njegovim bitovima. Također postoji i barem jedan bit koji govori je li prihvat prekida omogućen ili nije. Ti se bitovi automatski mijenjaju pri ulazu i izlazu iz sustavskog načina rada.

Pri prelasku u jezgreni način rada (u jednostavnom modelu računala):

- Privremeno se onemoguće daljnji prekidi
- Aktivira se sustavski način rada
- Aktivira se drugi sustavski registar kazaljke stoga, koji također postoji u procesoru, ali je odvojen od korisničkog stoga
- Pohranjuje se trenutni sadržaj programskog brojila i registra stanja na sustavski stog
- U registru stanja postavlja se zastavica koja označava sustavski način rada
- U programsko brojilo postavlja se adresa na kojoj počine potprogram za obradu prekida (adresa se nalazi u jezgrinom adresnom prostoru)

Prekidni signal u ovom slučaju djeluje kao sklopovski izazvani poziv potprograma. Kontekst korisničke dretve sprema se na sustavski stog, kako bi se mogao obnoviti prilikom izlaska iz prekidne funkcije. Svaka pojava prekida izaziva pohranjivanje konteksta korisničke dretve i vraćanje tog konteksta u procesor na kraju posluživanja prekida.

Može se ustanoviti da će procesor u opisanom načinu rada izvoditi dvije dretve:

- Jednu dretvu u korisničkom načinu rada
- Jednu dretvu u jezgrinom načinu rada procesora

Ovim se načinom korisnost procesora povećala na oko 98%, pri čemu je preostalih 2% obrada prekida i kućanski poslovi, odnosno spremanje i vraćanje

konteksta korisničke dretve u procesor.

2.3. Podsustav za prihvat prekida razvrstanih po prioritetima

Program za ovakav prihvat prekida mogao bi se zasnovati na sljedećim pretpostavkama:

- Neka se program sastoji od neprekidivih dijelova koji obavljaju kućanske poslove i prekidivih dijelova u kojima se obavlja obrada prekida
- Svi prekidi, bez obzira na prioritet, prihvaćaju se onda kada se program nalazi u prekidivom dijelu
- Ako je novoprispjeli prekid nižeg prioriteta, on se stavlja na listu čekanja, a ako ima viši prioritet, on se počinje izvoditi
- Kada završi obrada nekog prekida nastavlja se obrada sljedećih (onih koji su bili prekinuti ili onih koji su na listi čekanja)
- Ako ni jedan prekid više ne čeka na obradu, nastavlja se izvođenje korisničkog programa

U sustavkom dijelu spremničkog prostora uz sustavski stog smješta se i posebna struktura podataka:

- Varijabla `T_P` u koju se stavlja broj tekućeg prioriteta dretve koja se upravo izvodi
- Poretka zapisa `KON[N]` u koji se može pohranjivati cijeli kontekst pojedine dretve koja se upravo izvodi
- Poretka `K_Z[N]` u koji se pohranjuju kopije zastavica iz pristupnih sklopova (0 = ne čeka obradu prekida, 1 = čeka obradu prekida)

2.4. Sklopovka potpoora za ostvarenje višestrukog prekida

Praktički isto kao i `2.3.`. Još uvijek ima više prioriteta, ali postoji jedna bitna razlika. Ako nema sklopovske potpore, dolaskom svakog prekida mora se provjeravati kojeg je on prioriteta. Znači čak i ako se izvodi najviša razina, još uvijek treba imati kućanske poslove za provjeru. Kod sklopovske potpore se eliminiraju ti kućanski poslovi. Ako je stigao prekid nižeg prioriteta, obrada tretnunog prekida se samo nastavlja. Umjesto poziva programa i provjere svaki puta, sklop sada provjera umjesto procesora i šalje mu signal samo kada stigne neki veći prioritet.

2.5. Prekidi generirani unutar procesora

- Nenamjerni prekidi
  - dijeljenje s 0
  - adresiranje nepostojeće adrese
  - dekodiranje nepostojećeg operacijskog koda
- Namjerni prekidi
  - Procesor ima barem jednu instrukciju koja omogućuje prekide iz programa (programski prekid)

– Poziv jezgre (system call)

2.6. Prenošenje bloka znakova

Kako se nebi događao prekid kod prenosa svakog znaka, pristupnom sklopu dodijeljena su još 2 registra:

- Adresni registar AR u kojem će se nalaziti adresa u spremniku na koju ili s kojeg će se obaviti prijenos znaka
- Brojilo BR u kojem se nalazi broj znakova koje još treba prenijeti
- (Osim toga od prije već imaju i podatkovni registar PR i registar stanja RS)

Sad pristupni sklop može od procesora zatražiti sabirničke cikluse, onoliko koliko treba, pri čemu se procesor mora odreći tih nekoliko ciklusa kako bi se mogli prenijeti podaci. Tek kad se svi podaci (cijeli blok) prenesu dolazi do prekida procesora. Obrada tog prekida sastoji se od ponovne inicijalizacije AR i BR ako se želi nastaviti s prijenosom ili obustavljanjem prijenosa. Ovakav način prijenosa, u kojem procesor ne mora sudjelovati naziva se Direct Memory Access ili DMA.

2.7. Čvrsto povezani višeprocesorski sustav

Svaki procesor ima svoj lokalni spremnik kojem samo on može pristupiti. Također postoji i zajednički spremnik kojem mogu pristupiti svi procesori, ali kako mu žele pristupati preko iste sabirnice, dodan je dodjeljivač, koji na početku svakog sabirničkog ciklusa određuje koji će procesor imati pravo na sabirnicu, za vrijeme trajanja tog jednog ciklusa.

3. Međusobno isključivanje u višedretvenim sustavima

[!INFO] Pročitati knjigu samo, a tu si na kraj napisi pseudokodove za algoritme (dekker, petersonov i lampart)

4. Jezgra operacijskog sustava

4.1. Radno okruženje za izvođenje dretvi - jednostavi model jezgre

Postoje 3 vrste prekida, koji služe za pozivanje jezgrinih funkcija:

1. Sklopovski prekidi ulazno-izlaznih naprava
2. Programski prekidi koje izazivaju jezgre
3. Prekidi od sata -> Ovo je korisno da se recimo spriječi deadlock.

Prekidom se poziva jezgrina funkcija, a izlazak iz jezgre svodi se na pokretanje jedne od dretvi, pri čemu se procesor vraća u korisnički način rada. Sama jezgra operacijskog sustava sastoji se od jezgrinih funckija i struktura podataka jezgre.

Za svaku dretvu potrebno je u jezgri pohraniti sve važne podatke. Ti podaci smještaju se u strukturu koja se zove opisnik dretve. U opisinku su spremljeni podaci poput:

- Identifikatora procesa kojem pripada dretva
- Identifikatora dretve da se svaka dretva može razlikovati jedna od druge
- Stanje dretve (pasivna, aktivna, blokirana)
- Početna adresa dretvenog adresnog prostora, veličina prostora
- Adresa prve instrukcije
- Zadano kašnjenje
- Prostor za smještanje konteksta, kad dretva bude prekinuta s radom

Postoji lista postojećih dretvi, lista aktivnih dretvi i red pripravnih dretvi. Lista postojećih dretvi sadrži opisnike svih dretvi u adresnom prostoru nekog procesa kako bi se njihovi opisnici mogli pregledati kada je to potrebno. Primjetiti da se koriste dvije kazaljke. Prva služi za povezivanje opisnika u druge liste, a druga služi za pokazivanje na sljedeći opisnik u listi. Postoji i lista aktivna dretva, koja može imati samo jedan opisnik u jednoprocesorskom sustavu. U višeprocesorskom sustavu, svaki procesor ima svoju listu aktivna dretva.

Red pripravnih dretvi služi kako bi se mogla pustiti nova dretva u procesor, kad trenutna aktivna dretva izađe. Primjeti da se ovdje koristi red, odnosno, ona dretva koja je prva u redu biti će puštena u procesor, a nove dretve koje postaju pripravne stavljaju se na kraj reda. Zaglavlje reda sadrži i kazaljku na trenutni zadnji element u redu, kako bi umetanje nove pripravne dretve u red moglo biti odrađeno u O(1) vremenu.

Izlazak iz jezgre radi se sljedećim koracima, pri čemu je bitan redoslijed:

1. Premjetiti prvi opisnik iz `Pripravne_D` i listu `Aktivne_D`
2. Obnoviti kontekst (bez RS i PC) iz opisnika `Aktivna_D`
3. Vratiti se u prekinutu dretvu
    1. Obnavlja se registar stanja čime procesor
        1. Automatski prelazi u način rada prekinute dretve
        2. Omogućava se prekidanje ako je prekidanje bilo dozvoljeno u prekinutoj dretvi
        3. Aktivira se odgovarajući (user/kernel) registar kazaljke stoga i addr. prostor
    2. Obnavlja se sadržaj programskog brojila

Ukoliko je red pripravnih dretvi prazan, uvijek postoji jedna latentna dretva, koja može samo trošiti vrijeme procesora, a osigurava da procesor uvijek nešto radi.

Dretve tijekom svog izvođenja mogu biti blokirane čekajući na ispunjenje nekog uvjeta za njihovo daljnje napredovanje. U jednostavnom modelu jezgre, dretve mogu biti blokirane na 4 načina:

1. Čekanjem na binarnom semaforu
2. Čekanjem na općem semaforu

3. Čekanjem na istek zadanog intervala kašnjenja
4. Čekanjem na završetak ulazno-izlazne operacije

4.2. Binarni semafor

Sastoji se od varijable, recimo `Bsem[i].v` i kazaljke koja pokazuje na red opisnika dretvi koje nisu uspijele proći semafor. Vrijednost binarnog semafora može biti 0 ili 1, pri čemu vrijednost 0 znači da semafor nije prolazan, a 1 da je prolazan. Postoje i dvije funkcije, ispitati i postaviti binarni semafor. Ispitati čeka ukoliko je vrijednost 0, ili prolazi i postavlja vrijednost na 0. Postaviti podiže vrijednost na 1.

4.3. Opći semafor

Sastoji se od istih varijabla i funkcija kao i binarni semafor, ali vrijednost varijable kod općeg semafora može poprimiti vrijednosti veće od 1 (2, 3, 4….)

4.4. Čekanje na istek zadanog intervala kašnjenja

Postavlja se vrijednost zadanog kašnjenja u opisniku na neki višekratnk periode otkucaja sata. Opisnik dretve koji se želi odgoditi stavlja se u poseban red odgođene dretve, pri čemu je vrijednost kašnjenja u opisniku dretve postavljena relativno na prethodne dretve u redu. To omogućava da se vrijednost za kašnjenje uvijek mijenja samo u jednoj dretvi, i to onoj koja će prva biti gotova s čekanjem. (npr D1 kasni 10, a D2 35, zapisano bi bilo zag->D1=10->D2=25; Ako se pojavi D3 kasni 15, sad izgleda D1=10->D3=5->D2=20).

Nakon što prođe uvjet zbog kojeg je dretva bila blokirana, njen opisnik stavlja se u listu **pripravnih** dretvi. Dretva iz aktivnog stanja može izaći na nekoliko načina:

- Ako je dretva uspješno obavila svoj posao, pri čemu će pozvati jezgrinu funkciju *završiti*
- Ako dretva želi ispitati opći ili binarni semafor, ona će pozvati jezgrine funckije *ispitati* i ako su semafori neprolazni biti će blokirane
- Ako dretva samu sebe želi zakasniti, poziva jezgrinu funckiju *zakasniti_sebe* i njezin je opisnik prebaćen u red odgođenih dretvi
- Ako dretva želi obaviti ulazno-izlaznu operaciju preko naprave, biti će blokirana i njen će opisnik biti smješten u pripadni red
- Ako se dogodi neki sklopovski prekid, dretva je izbačena iz aktivnog stanja.

Funkcije za ispitivanje semaforu zovu dretve prije samog ulaza u kritični odsječak. Na taj način osiguravaju da uvijek samo jedna dretva ima pristup kritičnom odsječku (na primjer ako više dretvi želi modificirati vrijednost neke varijable).

5. Pseudokodovi

5.1. Dekkerov algoritam

- Međusobno sinkroniziranje samo 2 dretve
- Ovisi o početnoj vrijednosti varijable pravo

```
dretva(i) {
  dok je (1) {
    z[i] = 1;
    dok je (z[j] != 0) {
      ako je (pravo != i) {
        z[i] = 0;
        dok je (pravo != i); // Radno cekanje
        z[i] = 1;
      }
    }
    Kriticni odsjecak;
    pravo = j;
    z[i] = 0;
    Nekriticni odsjecak;
  }
}
```

5.2. Petersonov algoritam

- Daje prednost bržoj dretvi
- Ne ovisi o početnoj vrijednosti varijable pravo
- Također se koristi za sinkroniziranje samo 2 dretve

```
dretva(i) {
  dok je (1) {
    z[i] = 1;
    pravo = j;
    dok je ((pravo == j) && (z[j] == 1)); // Radno cekanje
    Kriticni odsjecak;
    z[i] = 0;
    Nekriticni odsjecak;
  }
}
```

5.3. Laportov algoritam

- Jedini od navedenih algoritama za sinkroniziranje koji može sinkronizirati više od 2 dretve

```
dretva(i) {
  ulaz[i] = 1;
  broj[i] = zadnji_broj + 1;
```

```
  zadnji_broj = broj[i];
  ulaz[i] = 0;
  za (j = 0; j < n; j++) {
    dok je (ulaz[j] == 1);
    dok je ((broj[j] != 0) && ((broj[j], j) < (broj[i], i)));
  }
  Kriticni odsjecak;
  broj[i] = 0;
  Nekriticni odsjecak;
}
```

5.4. Sklopovska potpora međusobnom isključivanju

U jednoprocesorskom sustavu

```
dok je (1) {
  onemoguciti prekidanje;
  Kriticni odsjecak;
  omoguciti prekidanje;
  Nekriticni odsjecak;
}
```

U višeprocesorskom sustavu:

- Ispitati i postaviti - TAS
- Zamijeniti - SWAP
- Dohvatiti i pocećati - Fetch and Add

Compare and swap:

```
CAS (adresa, usporedi, postavi) {
  // 1. sabirnicki ciklus
  procitano = dohvati_s_adrese(adresa);
  ako je (procitano == usporedi) {
    // 2. sabirnicki ciklus
    pohrani_na_adresu(adresa, postavi);
  }
}
```

5.5. Funkcije za binarni semafor

```
// Ispitati binarni semafor
  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;
    vratiti se u prekinutu dretvu;
  } inace {
```

```
    premjestiti opisnik iz reda Aktivna_D u red bsem[i];
    aktivirati prvu dretvu iz reda Pripravne_D;
  }
}

// Postaviti binarni semafor
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;
  } inace {
    bsem[i].v = 1;
  }
  aktivirati prvu dretvu iz reda Pripravne_D;
}
```

5.6. Funkcije za opći semafor

```
// Ispitati opci semafor
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;
    vratiti se u prekinutu dretvu;
  } inace {
    premjestiti opisnik iz reda Aktivna_D u red osem[j];
    aktivirati prvu dretvu iz reda Pripravne_D;
  }
}

// Postaviti opci semafor
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;
  } inace {
    osem[j].v = osem[j].v + 1;
  }
  aktivirati prvu dretvu iz reda Pripravne_D;
}
```