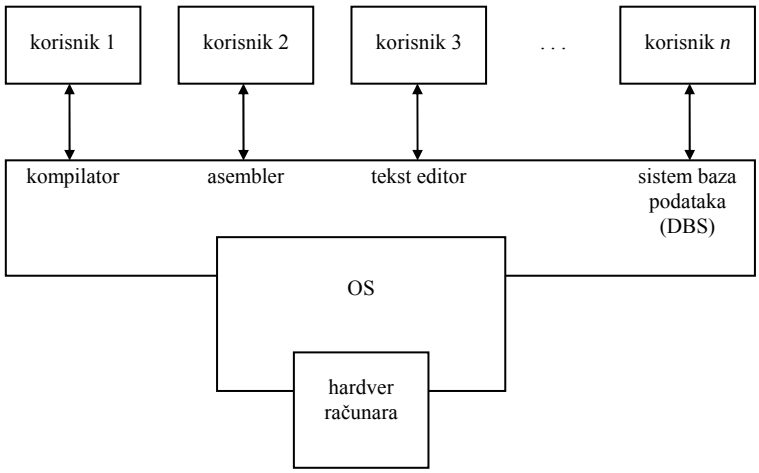
4. OPERATIVNI SISTEM

Operativni sistem (OS) je važan deo skoro svakog računarskog sistema. Računarski sistem (RS) se može grubo sagledati kroz sledeće četiri komponente: hardver, OS, aplikacioni programi i korisnici (slika 4.1).



Sl. 4.1. Blok šema računarskog sistema.

Hardver (CPU+memorija+U/I uređaji) predstavlja osnovni računarski resurs. Aplikacioni programi (kompilator, DBS, bankarski programi i dr.) definišu način na koji će se ovi resursi koristiti da bi se uspešno rešio problem korisnika. Postoji veći broj korisnika koji pokušavaju da reše različite probleme. Saglasno tome postoje i različiti aplikacioni programi. OS upravlja i kordiniše korišćenjem hardvera između aplikacionih programa različitih korisnika. OS je sličan vladi. Komponente RS su hardver, softver i podaci. OS obezbeđuje sredstvo za korektno korišćenje ovih resursa u toku rada OS-a. Slično vladi, OS ne obavlja neke korisne funkcije za sebe. On jednostavno obezbeđuje okruženje u okviru koga drugi programi mogu obavljati koristan posao. Drugim rečima, OS se može posmatrati kao deo koji dodeljuje resurse (alokator resursa).

4.1. Funkcije operativnog sistema

Globalno posmatrano, OS obavlja dva tipa osnovnih funkcija:

a) korisničke funkcije,
b) sistemske funkcije.

Korisničke funkcije obezbeđuju korisniku okruženje (virtuelnu mašinu) za izvršavanje programa. Specifične programske funkcije razlikuju se od jednog OS-a do drugog, ali u principu možemo identifikovati neke zajedničke, a to su:

i) **Upravljanje programom** - korisnik treba da je u stanju da manipuliše programom i podacima u okviru RS-a; na primer, kompilacijom i izvršenjem određenog programa, nad definisanim skupom podataka. Korisnik, shodno tome, koristi komandni jezik koji mu omogućava da upravlja virtuelnom mašinom.
ii) **U/I operacije** - Kako se U/I funkcije podržavaju na arhitekturnom nivou njihovo korišćenje može biti komplikovano i složeno, OS je taj koji obezbeđuje da U/I operacije budu jednostavne, moćnije i prihvatljivije za korisnika.
iii) **Manipulacije kod rada sa datotekama** - Fajl sistem se koristi za memorisanje programa i podataka. OS omogućava korisniku da pristupa i manipuliše memorisanim informacijama koristeći simbolička imena, a ne fizičke lokacije zapisa na medijumu (disku ili traci).

Sistemske funkcije - moguće je identifikovati sledeće sistemske funkcije:

a) **Upravljanje memorijom** - ako virtuelna mašina poseduje različit (često veći) adresni prostor od obima fizičke memorije koji je instaliran u realnoj memoriji, jedna od funkcija OS-a je da upravlja ovom virtuelnom memorijom.
b) **Zaštita** - programima treba garantovati nivo privatnosti i nepostojanje interferencije, ali u isto vreme treba omogućiti da oni između sebe komuniciraju. Zbog toga interakcija između programa korisnika i između programa korisnika i OS, treba da bude nadgledana, upravljana i zaštićena.
c) **Upravljanje resursima i planiranje** - kada se nekoliko programa izvršava konkurentno, resursi (kao što su CPU, memorija i U/I uređaji) se moraju dodeljivati svakom od programa, ali tako da se obezbedi zaštita od konflikta. Šta više, planiranje izvršenja treba da se izvede tako da se postigne optimalno korišćenje ovih resursa.
d) **Vođenje evidencije** - OS treba da vodi računa o tome koliko puta je određeni resurs korišćen od strane programa.

4.2. Tipovi operativinih sistema

OS opšte namene mogu se podeliti na: serijske sisteme sa paketnom (batch) obradom, multiprogramske sisteme i sisteme sa deljenjem procesorskog vremena (time-sharing).

Kod "batch" sistema programi se prvo čitaju sa čitača kartica, nakon toga izvršavaju i na kraju štampaju. Ove aktivnosti se ponavljaju u toku izvršenja svakog programa.

Kod multiprogramskih OS-a dozvoljeno je preklapanje izvršenja aktivnosti tipa unošenje podataka, obrada i generisanje izlaza. Kada CPU treba da čeka neku U/I operaciju (pristup disku ili drugu U/I operaciju) on se prebacuje na izvršenje drugog programa koji nije blokiran zbog U/I. Na ovaj način se dobija veći stepen iskorišćenja CPU-a.

"Time-sharing" je oblik multiprogramiranja kod koga postoji interakcija između korisnika i programa (pomoću terminala), dok se program izvršava. Ovo se ostvaruje izvršenjem svakog korisničkog programa za kratak vremenski interval, nakon čega se izvršava drugi korisnički program. Na ovaj način svaki korisnik dobija utisak kao da je jedini korisnik sistema.

4.3. Model procesa

Moderni računari mogu obavljati nekoliko zadataka u isto vreme. Na primer, "time-sharing" sistem može da komunicira sa korisnikom, da obavlja terminalnu U/I aktivnost, a da u isto vreme čita/upisuje podatke sa/na disku. Kod multiprogramskog režima rada ili kod "time-sharing" on može da komutira sa jednog programa na nekoliko drugih više puta u sekundi, pa se stvara utisak da postoji virtuelni paralelizam. Osnovni zadatak OS-a je da upravlja ovim virtuelnim paralelnim aktivnostima koje mogu biti veoma komplikovane. Godinama su projektanti OS-a razvijali model koji se naziva *model procesa* pomoću koga se manipuliše kompleksnošću pomenutog paralelizma.

4.3.1. Definicija procesa

Model procesa predstavlja strukturni način opisivanja aktivnosti OS-a. Celokupni softver, uključujući OS, organizovan u vidu većeg broja programa, kada se aktivira, formira proces. Ovo znači da procesi mogu biti u nekom aktivnom stanju ili mogu biti izbačeni iz sistema.

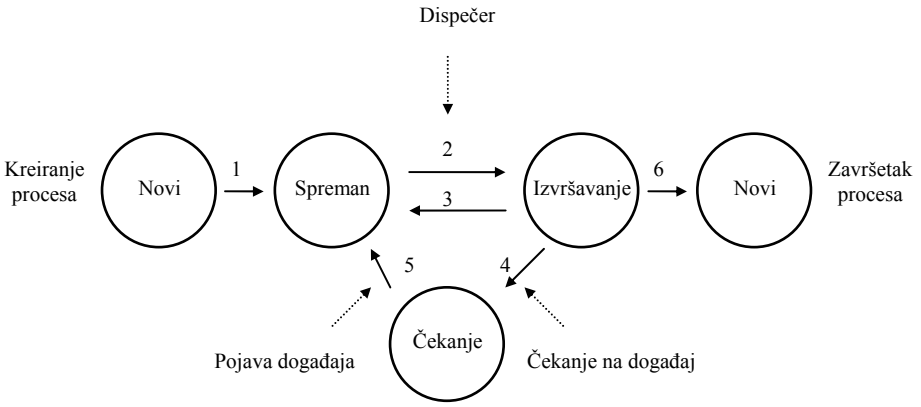
Koncepcijski posmatrano, svaki proces se može izvršavati na posebnom procesoru. Ali u praksi, kada se govori o multiprogramskom režimu rada, to znači da veći broj procesa može da deli jedan procesor. Shodno tome, OS je taj koji obavlja planiranje izvršenja procesa sa ciljem da odredi kada treba zaustaviti izvršenje jednog procesa, a početi sa opsluživanjem drugog.

OS zasnovan na modelu procesa obezbeđuje način za kreiranje i potiskivanje procesa. Inicijalno, jedan proces startuje sa izvršenjem. Ovaj, inicijalni, proces može da produži sa izvršenjem sam za sebe, ili da kreira nove, nezavisne procese putem mehanizma sistemskog poziva. Kada proces kreira novi proces, početni proces (proces roditelja) može da produži sa izvršenjem ili može biti suspendovan, dok će novi proces (nazvan proces naslednik), početi sa izvršenjem konkurentno, bez bilo kakvog efekta na drugi proces, i pored toga što oba procesa pripadaju istom delu programskog koda. Ova kostatacija čini da je notacija procesa različita od notacije potprograma.

4.3.2. Stanje procesa

U toku izvršenja, nivo aktivnosti procesa, nazvan stanje, se menja. Stanje procesa se definiše pomoću njegove tekuće aktivnosti. Izvršenje procesa čini alternativna sekvenca CPU-ove i U/I aktivnosti, koja počinje i završava sa CPU-ovom aktivnošću. Kao što je prikazano na slici 4.2, razlikuje se pet stanja procesa i šest prelaza.

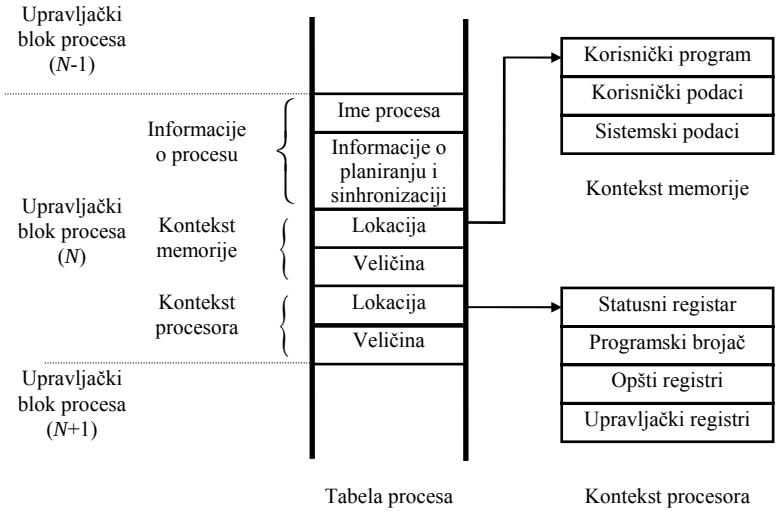
Kod nekih OS-a postoje i druga stanja, kao što je suspendovano stanje, koje se koristi za privremeno zaustavljanje procesa.



Sl. 4.2. Stanja procesa.

4.3.3. Kontekst procesa

Kontekst procesa se definiše kao informacija koja je neophodna da se u potpunosti specificira njegovo tekuće stanje. Ovo uključuje svu informaciju koju je potrebno memorisati kada proces napusti stanje izvršenja (*running state*) i mora da se obnovi kada ponovo uđe u stanje izvršenja. Kao što je prikazano na slici 4.3, informacija o kontekstu procesa se može podeliti na sledeće delove (u zavisnosti od toga kada ona mora biti rezidentna):



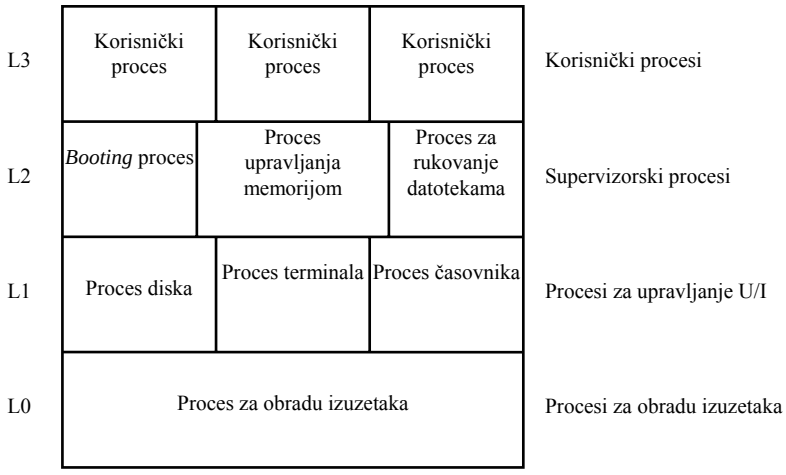
Sl. 4.3. Kontekst procesa.

1) **Upravljački blok procesa (PCB):** Ovaj blok sadrži informaciju koja je smeštena u glavnoj memoriji u toku egzistencije procesa (stanja *New*, *Waiting*, *Ready* i *Running*). Informacija smeštena u PCB je potrebna OS-u radi upravljanja procesom. PCB-ovi svih procesa u sistemu smeštaju se u velikoj tabeli koja se zove tabela procesa (PT). Svaki PCB, minimalno, mora da sadrži sledeću informaciju:
   - identifikacioni broj procesa,
   - informaciju o planiranju i sinhronizaciji (odnosi se na stanje i prioritet procesa i koliko je dugo taj proces bio u tom stanju),
   - lokaciju i obim memorijskog konteksta, i CPU-ovog konteksta procesa, koji kod alokacije i loadovanja procesa u memoriji obezbeđuje kontekst tom procesu u trenutku kada se vrši njegovo planiranje radi izvršenja.
2) **Kontekst memorije** - je deo za program i podatke koji pripada ontekstu procesa. Ova informacija treba da bude rezidentna u memoriji samo kada je proces u stanju "running" ili "ready", i obezbeđuje da prelaz 2 (slika 4.2) bude veoma brz.

3) **Kontekst procesora** - je deo konteksta procesa koji se čuva u CPU-ovim registrima. Ovi registri su integralni deo CPU-a i obezbeđuju veoma brzu obradu. Kod najvećeg broja računara, kontekst procesora čine sledeća četiri dela;
   - statusni registar (SR),
   - programski brojač (PC),
   - registri opšte namene, i
   - upravljački registri.

4.3.4. Nivoi procesa

Proces je moguće struktuirati. Proces koji koristi rezultat, ili zavisi od rezultata drugog procesa, pripada hijerarhijski višem nivou. Slični procesi se postavljaju na istom nivou. Struktuirana prezentacija računarskog sistema prikazan je na slici 4.4.



Sl. 4.4. Nivoi procesa u računarskom sistemu.

Svaki nivo koji se nalazi iznad prethodnih nivoa, označava da su opsluživanja koja propadaju nižim nivoima dostupna procesima koji se izvršavaju na višim nivoima. Razlikujemo četiri nivoa procesa:

1. **Korisnički procesi**: obavljaju izvršenje korisničkih programa, koristeći usluge supervizorskih procesa. U daljem tekstu ih nećemo razmatrati.
2. **Supervizorski procesi**: izvršavaju manje kritične fukciije OS-a, kao što su manipulisanje (upravljanje) fajlovima (datotekama), upravljanje memorijom i planiranjem.
3. **Procesi za U/I obradu**: izvršavaju vremensko-kritične funkcije. Ovi procesi se tipično odazivaju na prekide i obavljaju "device driver" funkcije, koje se sastoje od nekoliko U/I operacija nižeg nivoa.
4. **Procesi za rukovanje izuzecima**: izvršavaju akcije pomoću kojih se očuvava integritet sistema i dozvoljava u određenoj meri tolerantnost na greške. Greške i izuzeci koji se detektuju u toku izvršenja procesa na višim nivoima se obrađuju na ovom nivou.

OS obavlja veliki broj funkcija, od kojih svaka može da se obavlja od strane jednog ili većeg broja procesa (na nivoima L2, L1 i L0 na slici 4.4). Najvažnije supervizorske funkcije su: upravljanje procesom, U/I obrada, i upravljanje memorijom.

4.3.5. Upravljanje procesom

Funkciju upravljanje procesom čini skup procesa koji mogu pripadati nekom od različitih nivoa (slika 4.4). Ova funkcija obavlja sledeće zadatke:
- kreiranje i ažuriranje strukture podataka procesa,
- kreiranje i izbacivanje (ukidanje) procesa,
- zaštita procesa,
- sinhronizacija procesa,
- planiranje procesa,
- komutacija procesa.

4.3.6. Opsluživanje U/I

Obrada U/I, kao što su diskovi, terminali ili satni mehanizam su deo supervizorskih funkcija iz sledećih razloga:

- Hardver najvećeg dela U/I uređaja zahteva kompleksan softver za njegovo upravljanje i korišćenje. OS krije ove detalje u rutinama "device driver"-a preko kojih korisnički procesi mogu prenositi podatke, a time i vrše upravljanje ovim uređajima.
- OS treba da obezbedi korisniku konzistentan, uniforman i fleksibilan interfejs za sve uređaje, čineći pri tome da detalji koji su karakteristični za upravljanje svakog specifičnog uređaja budu nevidljivi. Ovo omogućava korisniku da napiše programe pomoću kojih se obraćanje uređajima vrši na osnovu imena (na primer preko fajl sistema), a da se istovremeno izvršavaju operacije visokog nivoa bez poznavanja specifičnih detalja koji se odnose na rad uređaja.
- Pošto se uređaji mogu deliti od strane većeg broja procesa oni se moraju zaštititi i dodeliti od strane OS-a, sa ciljem da se ostvari siguran i ispravan pristup njima.

4.3.7. Upravljanje memorijom

Pored U/I uređaja, glavna memorija je drugi važan resurs sa kojim OS treba da upravlja. Upravljanjem memorije čuva se trag o lokaciji i obimu dostupne memorije. Memorija se dodeljuje kada je proces zahteva, a određuje njena fizička lokacija kada je neophodno. Računarski sistem često obezbeđuje virtuelan memorijski prostor za svaki proces. Zbog toga, OS mora da deli memoriju između svih onih procesa koji žele da je koriste.

4.3.8. Zaštita procesa

Kao što smo već uočili, veći broj procesa (kako korisničkih tako i supervizorskih) može biti prisutan u računarskom sistemu. Bez određenog oblika zaštite, integritet rada sistema se ne može garantovati, jer korisnički proces, na primer, može imati pristup informaciji o planiranju izvršenja procesa pa da na taj način spreči izvršenje drugih procesa. Zaštita svih procesa (bilo da su oni korisnički ili supervizorski) je neophodna, i ona mora da vodi računa o:

- **sigurnosti** - prevenciji sistema od kraha usled nenamernih ili malicioznih akcija ili operacija,
- **privatnosti** - zaštiti da neovlašćeni korisnici pristupaju podacima i programima koji im nisu vidljivi.

Realizaciju zaštite procesa komplikuje činjenica da se procesi ne izvršavaju u izolaciji; procesi obavljaju funkcije za druge procese, pa zbog toga oni između sebe komuniciraju.

Zaštita se može izvesti na nekoliko načina od kojih su najkarakterističniji:

1) zaštita na HLL,
2) zaštita na arhitekturnom nivou,
3) zaštita domena zasnovanih na memorisjkom prostoru, i
4) zaštita domena zasnovanih na korisničko definisanim objektima.

U daljem tekstu ukazaćemo na osnovne osobine ovih načina zaštite u kratkim crtama.

1. Zaštita na HLL

Ovaj tip zaštite postoji kod mašina iz serije Burroughs B6000. Svi programi su pisani na HLL jeziku sličnom ALGOL-u, koji je siguran sa stanovišta operacija, korišćenja i adresiranja memorije (ne koriste se pokazivači). Procesi su sigurni jer jezik ne sadrži nesigurne konstrukcije. Garantovana je i privatnost, jer se upravljanje memorijom izvodi za svaki objekat. Ovaj metod se me može primeniti generalno, jer veći broj jezika, kao što je Pascal, koristi konstrukcije koje će dovesti do nesigurnosti u radu (na primer korišćenje pokazivača).

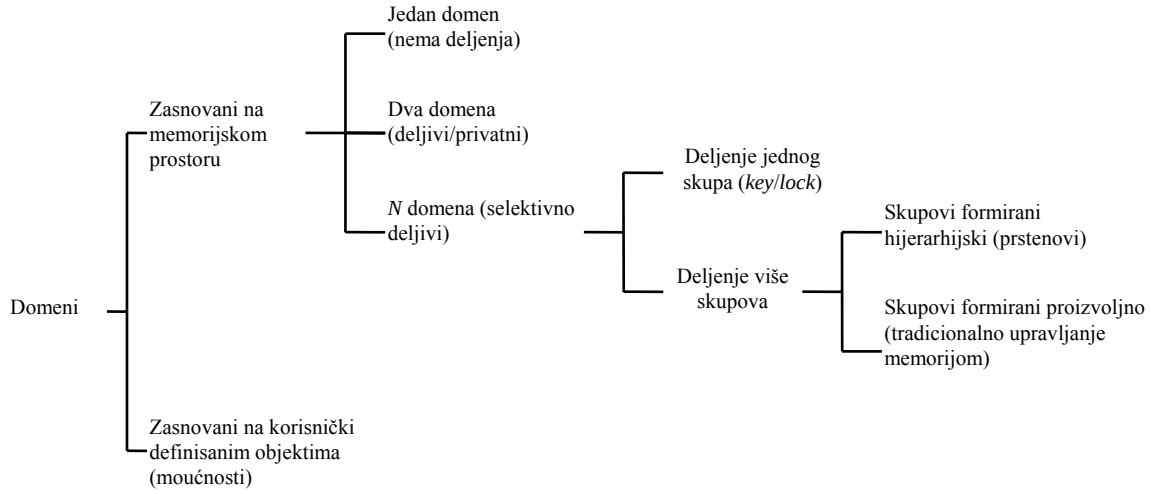
2. Zaštita na arhitekturnom nivou

Kod ovog tipa zaštite u toku izvršenja programa vrši se detekcija ilegalnih operacija, a shodno tome obavlja se prevencija da se one ne izvrše. Nelegalnost može biti uzrokovana nelegalnom komponentom u bilo kom delu (polju) instrukcije u kojoj se vrši kodiranje: opkoda, operamdskog ili prostora za podatke. Kada se detektuje, izvršenje instrukcije kojoj komponenta pripada se štiti, a njena pojava se signalizira programu za obradu izuzetaka preko mehanizma sličnom prekidu, a zove se izuzetak.

a) **zaštita preko opkod prostora** - izvšenje nekih instrukcija dozvoljeno je samo procesima OS-a. Ove instrukcije se zovu privilegovane instrukcije (instrukcije za: manipulisanje sa prekidima i izuzecima, komutaciju procesa, upravljanje memorijom i U/I). Da bi odredio kada je izvršenje privilegovanih instrukcija dozvoljeno ili ne, računar mora da radi u (najmanje) dva načina rada: supervizorski način rada i korisnički način rada. Instrukcije koje mogu da se izvršavaju samo u supervizorskom načinu rada zovu se privilegovane (specifikacija

privilegovanosti vrši se preko opkoda). Sve ostale instrukcije se izvršavaju u oba načina rada. Obično tekući način rada se naznačava pomoću određenog bita u registru PSW.

b) **zaštita preko prostora operanada** - operand može biti nelegalan ako je privilegovan, pristupa mu se u korisničkom načinu rada (na primer, specifikacija registra SR u korisničkom načinu rada je nelegalna), ili je adresa operanda van domena. Upravljanje memorijom je mehanizam kojim se garantuje integritet operanda. Ovaj mehanizam garantuje procesu da ne postoji neželjeni uticaj od strane drugih procesa, i potencijalno od samog sebe, ako postoji veći broj domena po jednom procesu. Domen je "objekat" koji se štiti, a čini ga memorijska oblast kao što je stranica ili korisnički definisan objekat kao što je polje. Na slici 4.5 prikazani su tipovi za zaštitu memorije, koji se baziraju na tome šta određuje domen, broj domena i deljivost domena.



Sl. 4.5. Tipovi zaštite memorije.

c) **zaštita domena preko prostora za podatke** - ovaj tip zaštite zahteva da je svakom podatku, u prostoru za podatke, prdruženo "tag" polje pomoću koga se specificira kom domenu taj podatak pripada. Ovaj način, i pored toga što je efikasan, zahteva povećanje kapaciteta memorije pa je sa te tačke gledišta neefikasan.

3. Zaštita domena zasnovana na memorijskom prostoru

Ovo je najtradicionalniji način izvođenja zaštite memorije. Domeni se specificiraju u zavisnosti od adresnog opsega, i formiranja (kreiranja) podopsega u ukupnom adresnom opsegu. Domeni nemaju direktan međusobni odnos sa korišćenim objektima. Grupa programa i objekata podataka se smešta u jedan domen, a to čini da zaštitni mehanizam bude veoma grub, što znači da se samo grupa objekata, a ne individualni objekti mogu zaštititi ili deliti. Prednost ove šeme ogleda se u njenoj implementaciji, jer mora da postoji po nekoliko domena, koji zahtevaju jedinstvenu zaštitu. U zavisnosti od broja domena, i kako se oni specificiraju, razlikujemo sledeće šeme zaštite (slika 4.5):

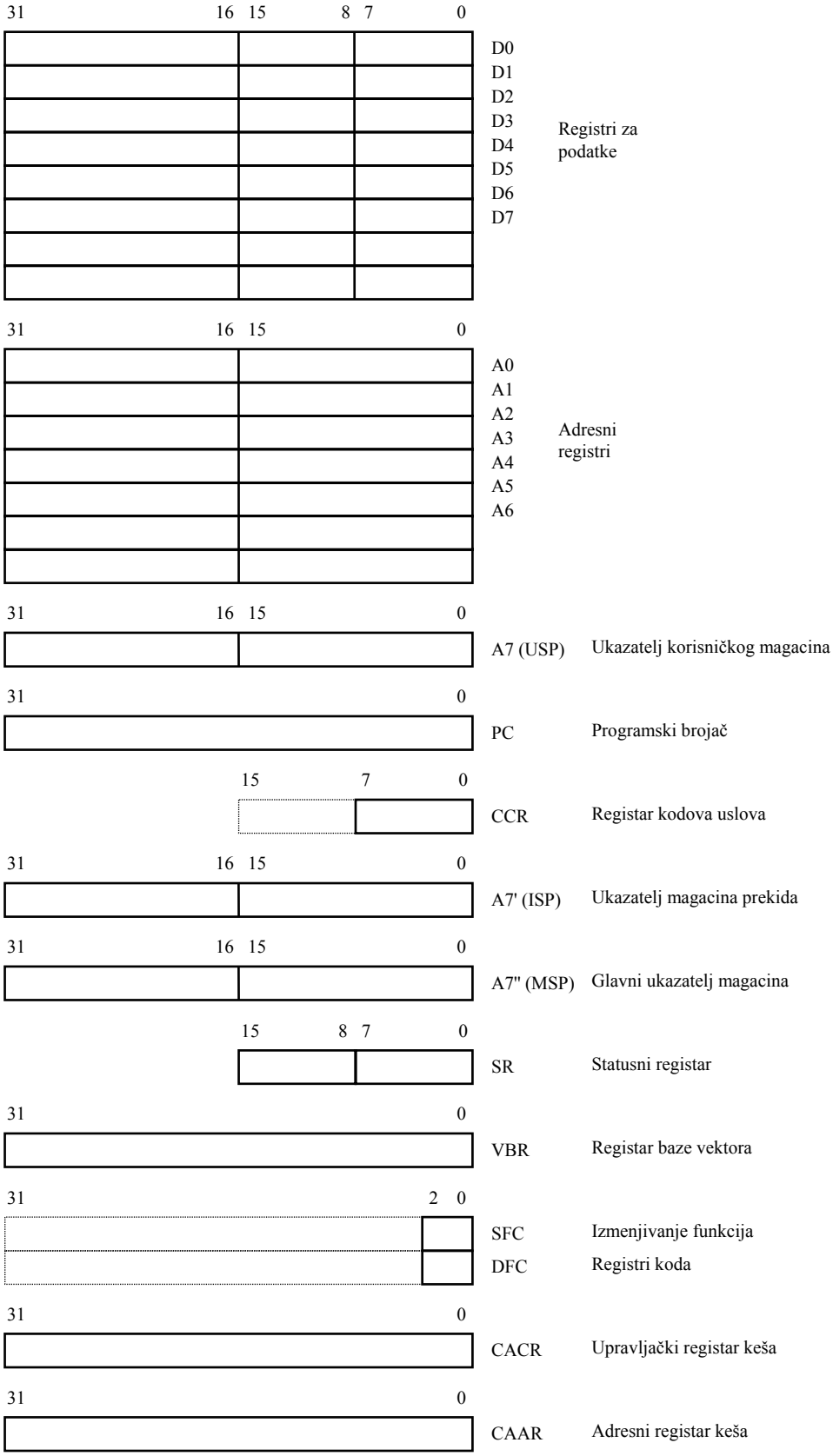
1) **jedinstveni domen (privatni)** je najjednostavniji oblik zaštite, jer ne dozvoljava deobu programa i podataka. Svakom procesu je dat jedinstveni (virtuelni) memorijski prostor kome se drugi procesi ne mogu obraćati, tako da je on za taj proces u strogom smislu privatan. Ovim načinom zaštite ne može se rešiti problem koji se javlja kada dva procesa dele programe i podatke.
2) **dva domena (deljivi/privatni)** je kompleksnija šema zaštite koja se dobija sledećim ograničenjem: Procesi mogu pristupati svojim privatnim domenima i jedinstvenom deljivom domenu, koji je dostupan svim procesima. Biblioteke tipa "Public" (matematička izračunavanja) se mogu smestiti u deljivom domenu.
3) **N domena (selektivna deljivost)** - može se izvesti na dva načina:
   - deobom jedinstvenog skupa domena,
   - deobom većeg skupa domena.

Detaljnija analiza ovog problema izlazi iz okvira ovog predmeta pa se zbog toga nećemo upuštati u dalja objašnjenja.

4. Zaštita domena koja se zasniva na korisničko definisanim objektima

Umesto da se specifikacija domena vrši u zavisnosti od prostora ona se može vršiti u zavisnosti od objekata. Objekti mogu biti programski objekti (procedure ili procesi) i/ili objekti podataka. Kako su programi sačinjeni od objekata definisanih na programskom nivou, svaki program može specificirati svoj sopstveni domen na bilo kom nivou granularnosti. Domen može da se sastoji od jedne promenljive tako da je zaštita i/ili deljivost na tom

nivou granularnosti moguća. Identifikacija domena se izvodi preko opisivača objekata i prava pristupa objektu (na primer, moguće je samo-čitanje, ili samo-izvršenje).



Sl. 4.6. (a) Opšti programski model za MC68020. (b) supervizorski dodatak programskom modelu za MC68020.

4.4. Supervizorski i korisnički način rada kod MC68020

Kao što smo već ranije ukazali kod MC68020 postoje dva načina rada:

a) supervizorski, i
b) korisnički.

Supervizorski način rada predstavlja viši nivo privilegije. Program u ovom načinu rada može da izvršava bilo koju instrukciju i da menja statusni registar i druge specijalne registre. U korisničkom načinu rada broj instrukcija koje se izvršavaju je ograničen.

Kao što je prikazano u Tabeli 4.1, u supervizorski način rada ulazi se kada CPU prepozna bilo koji tip izuzetka. Prelazak iz supervizorskog u korisnički način rada vrši se modifikacijom statusnog bita, (SR)[13], (pripada statusnom registru CPU-a). Kada je (SR)[13]={1} CPU radi u supervizorskom načinu rada, a kada je (SR)[13]={0} CPU radu u korisničkom.

Tab. 4.1.

	Supervizorski način rada	Korisnički način rada
U način rada se ulazi	Prepoznavanjem trapa, reseta ili prekida	Brisanjem statusnog bita ″S″
Sistemski pokazivač magacina	Supervizorski pokazivač magacina	Korisnički pokazivač magacina (A7)
Drugi pokazivači magacina	Korisnički pokazivač magacina (USP) i registri A0-A6	Registri A0-A6
Raspoloživi statusni bitovi		
Čitanje	C, V, Z, N, X, I_0 - I_2, S, T0, T1	C, V, Z, N, X
Upis	C, V, Z, N, X, I_0 - I_2, S, T0, T1	C, V, Z, N, X
Raspoložive instrukcije	Sve, uključujući i instrukcije za upravljanje sistemom	Sve osim onih koje rade u supervizorskom načinu rada
Registri specijalne namene	CAAR, CACR, DFC, SFC, VBR	-

Napomene:

1. Adresni registri A0-A6 se mogu koristiti kao pokazivači privatnih magacina u programima koji se izvršavaju u oba načina rada.
2. Registri specijalne namene upravljaju keš memorijom (CAAR, CACR), signalnim linijama koda funkcije (DFC, SFC) i lokacijom vektorske tabele (VBR). Ovi registri su objašnjeni u odgovarajućem odeljku.

Na slici 4.6a prikazan je opšti programski model mikroprocesora MC68020. Bilo kom od ovih registara se možemo obratiti pomoću naredbi na asemblerskom jeziku (u korisničkom i supervizorskom načinu rada). Dodatni registri dostupni samo u supervizorskom načinu rada prikazani su na slici 4.6b.

4.4.1. Instrukcije za upravljanje sistemom

Kod MC68020 postoji grupa instrukcija koja se zove instrukcije za upravljanje sistemom. Ovoj grupi pripadaju privilegovane instrukcije koje se koriste za dinamičku promenu režima rada računara. U Tabeli 4.2 dat je pregled ovih instrukcija.

Modifikacija statusnog registra

U Tabeli 4.3. prikazane su dostupne instrukcije u supervizorskom načinu rada koje mogu menjati sadržaj statusnog registra.

Manipulacija USP-om (User Stack Pointer)

Program koji se izvršava u supervizorskom načinu rada može da memoriše, obnovi i promeni sadržaj USP-a. Privilegovanom instrukcijom

MOVE.L USP,<An>

kopira se (USP) u An. Suprotan prenos ima oblik

MOVE.L <An>,USP

Manipulacija CCR-om

Instrukcijama prikazanim u Tabeli 4.4 dostupna su oba načina rada (supervizorski i korisnički). Logičke instrukcije omogućavaju da se sadržaj CCR-a modifikuje koristeći neposrednu 8-bitnu vrednost. Na primer, instrukcijom

ORI.B #1,CCR

postavlja se bit prenosa C={1} a ne menjaju se vrednosti ostalih bitova koji ukazuju na neki uslov.

Celokupni sadržaj CCR-a se može modifikovati izvršenjem instrukcije

MOVE.W <EA>,CCR

u toku čijeg izvršenja (<EA>)[7:0] sadrži nove bitove uslova za CCR. Operacija zahteva specifikaciju operanada tipa reč, ali se njenim izvršenjem koristi samo LS bajt za ažuriranje markera uslova.

Tab. 4.2. Instrukcije za upravljanje sistemom.

Instrukcija	Sintaksa operanda	Veličina operanda	Efekat
			Privilegovana
ANDI	#<data>,SR	16	Neposredna vrednost $\wedge$ SR $\rightarrow$ SR
EORI	#<data>,SR	16	Neposredna vrednost $\oplus$ SR $\rightarrow$ SR
MOVE	<EA>,SR	16	Izvor $\rightarrow$ SR
	SR,<EA>	16	SR $\rightarrow$ Odredište
MOVE	USP,An	32	USP $\rightarrow$ An
	An,USP	32	An $\rightarrow$ USP
MOVEC	Rc,Rn	32	Rc $\rightarrow$ Rn
	Rn, Rc	32	Rn $\rightarrow$ Rc
MOVES	Rn,<EA>	8, 16, 32	Rn $\rightarrow$ odredište koristeći DFC
	<EA>,Rn		Izvor koristeći SFC $\rightarrow$ Rn
ORI	#<data>,SR	16	Neposredna vrednost $\vee$ SR $\rightarrow$ SR
RESET	Nema	Nema	Postavlja $\overline{\text{RESET}}$ linijiu
RTE	Nema	Nema	((SP)) $\rightarrow$ SR; (SP) + 2 $\rightarrow$ (SP); ((SP)) $\rightarrow$ PC; (SP) + 6 $\rightarrow$ (SP); obnavlja magacin prema formatu
STOP	#<data>	16	Neposredni podataka $\rightarrow$ SR; STOP

Napomene:
1. $\wedge$ = logičko I.
2. $\vee$ = logičko ILI.
3. $\oplus$ = isključivo ILI.
4. Rc je proizvoljni registar specijalne namene ili ukazatelj korisničkog magacina (CAAR, CACR, DFC, ISP, MSP, SFC, USP, VBR).
5. Rn je proizvoljni adresni registar ili registar za podatke.

Tab. 4.3. Instrukcije za izmenu stanja procesora.

Sintaksa		Efekat
ANDI.W	#<d_{16}>,SR	(SR) $\leftarrow$ (SR) AND <d_{16}>
EORI.W	#<d_{16}>,SR	(SR) $\leftarrow$ (SR) EOR <d_{16}>
MOVE.W	<EA>,SR	(SR) $\leftarrow$ (EA)
ORI.W	#<d_{16}>,SR	(SR) $\leftarrow$ (SR) OR <d_{16}>

Napomene:
1. Sve instrukcije su privilegovane.
2. MOVE u SR zahteva adresni način rada za podatke za <EA>. Prema tome, svi adresni načini rada osim direktnog adresno registarskog su dozvoljeni za izvorni operand.
3. Sadržaj SR se čita pomoću MOVE.W SR,<EA> instrukcije. Odredište se može definisati adresnim načinom rada za podatke, tj. zabranjeni su adresno registarski direktni i PC relativni način rada.

Instrukcija RTE

RTE (Return from Exception) je privilegovana instrukcija i koristi se za punjenje SR-a i PC-a sa vrednostima koje su smeštene u supervizorskom magacinu. Aktivnost instrukcije RTE (slika 4.7) je sledeća:

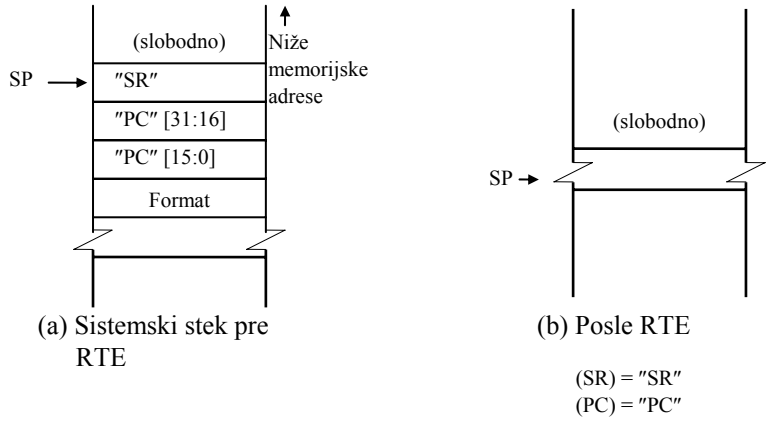
a) Load (SR)

(SR)[W] ← ((SSP)) ; POP(SR)
(SSP) ← (SSP)+2 ; ukaži na (PC)

b) Load (PC)

(PC)[L] ← ((SSP)) ; POP(PC)
(SSP) ← (SSP)+6

c) obnovi vrednost internih registara sa informacijom iz magacina.



(a) Sistemski stek pre RTE

(b) Posle RTE

Napomena: ″SR″ i ″PC″ su vrednosti koje zamenjuju tekući (SR) i (PC), respektivno.

Sl. 4.7. Efekat instrukcije RTE.

RTE se najčešće koristi kao zadnja instrukcija u rutini za obradu izuzetka. RTE se takođe koristi za prenos upravljanja u korisničkom načinu rada u toku izvršenja procedure za inicijalizaciju sistema.

Instrukcija RESET

Ova instrukcija je privilegovana i koristi se za resetovanje spoljnih interfejs kola u toku inicijalizacije sistema. Izvršenje ove instrukcije ima za efekat aktiviranje signalne linije CPU-a koja se koristi kao indikacija spoljnim kolima, tj. da ukaže da CPU zahteva inicijalizaciju odgovarajućeg interfejsa.

Instrukcija MOVEC i registri specijalne namene

MC68020 ima ugrađen veći broj registara specijalne namene koji su deo supervizorskog programskog modula (slika 4.6). Sadržajem ovih registara može se manipulisati privilegovanom instrukcijom MOVEC definisanom u Tabeli 4.4. Instrukcijom MOVEC vrši se prenos vrednosti između An ili Dn registra i registra specijalne namene.

Tab. 4.4. Kopiranje upravljačkog registra.

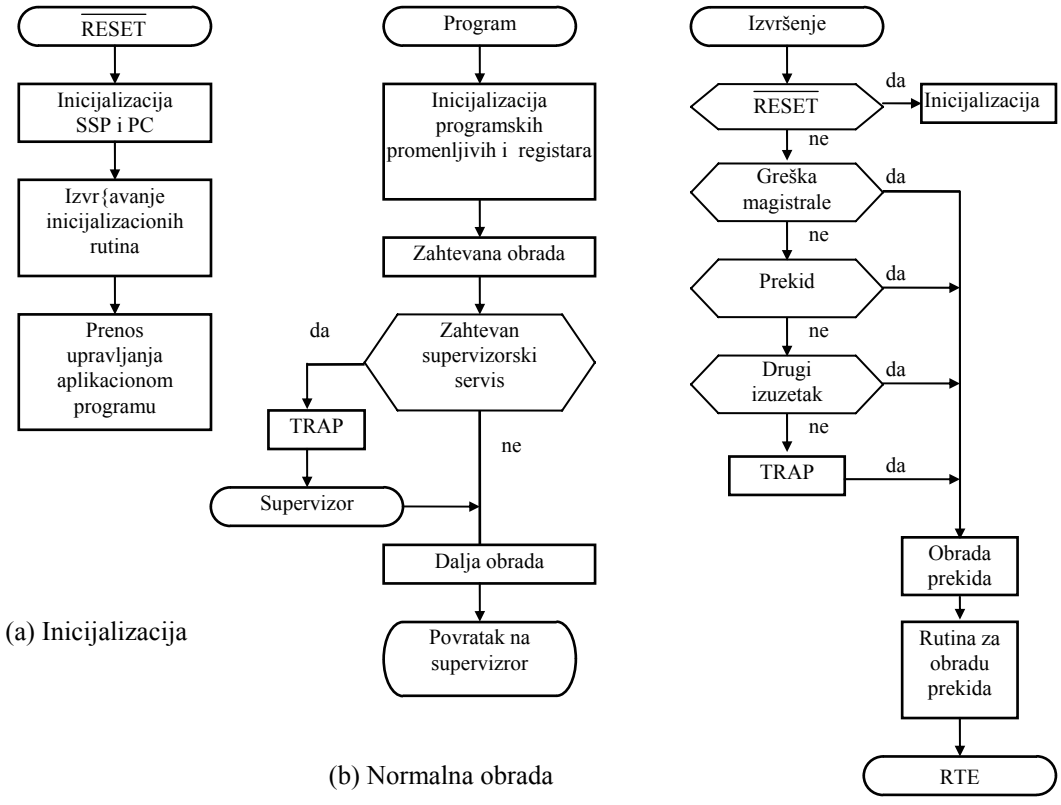
Sintaksa	Efekat
MOVEC <Rc>,<Rn>	(<Rn>) ← (<Rc>)
MOVEC <Rn>,<Rc>	(<Rc>) ← (<Rn>)

Napomene:
1. <Rn> je proizvoljan <An> ili <Dn>.
2. <Rc> je jedan od registara specijalne namene ili USP.
3. Sve operacije su 32-bitne.

Registri mikroprocesora MC68020 koji su naznačeni kao registri specijalne namene ili upravljački registri prikazani su u Tabeli 4.5.

Tab. 4.5. Registri specijalne namene.

Tip i mnemonik	Definicija
Upravljanje kešom	
CAAR	Adresni registar keša
CACR	Upravljački registar keša
Linije koda funkcije	
DFC	Odredišni kod funkcije
SFC	Izvorni kod funkcije
Ukazatelj supervizorskog magacina	
ISP	Ukazatelj magacina prekida
MSP	Ukazatelj glavnog magacina
Ukazatelj korisničkog magacina	
USP	Ukazatelj korisničkog magacina
Bazni registar vektora	
VBR	Bazni registar vektora



(a) Inicijalizacija

(b) Normalna obrada

(c) Izuzetak

Sl. 4.8.

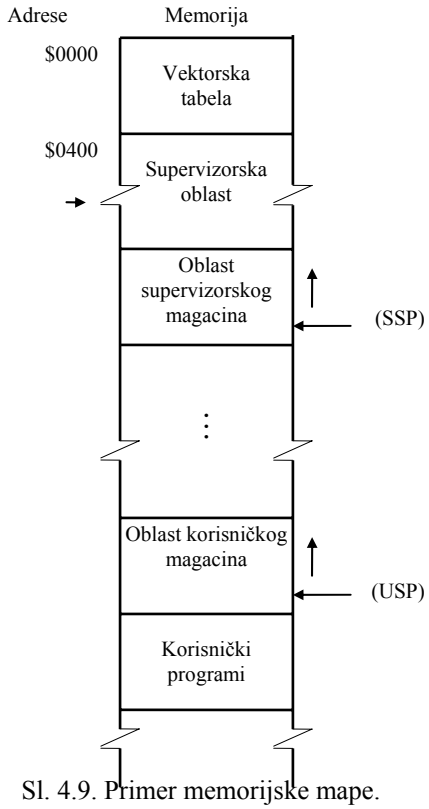
4.4.2. Opis rada sistema zasnovanog na MC68020

Mikroprocesor MC68020, kao što je prikazano na slici 4.8, može da se nalazi u jednom od sledeća tri procesna stanja: normalno, izuzetno ili zastoj.

Inicijalizacija sistema

Nakon uključenja sistema za napajanje, aktivira se sekvenca reset (slika 4.8a) pomoću koje se vrši inicijalizacija sistema. Sekvenca se inicijalizira spoljnim kolom koje aktivira $\overline{\text{RESET}}$ signalnu liniju za inicijalizaciju CPU-a. Inicijalizaciona sekvenca ima za cilj da postavi sistem u poznato (određeno) stanje ili uslov, pre nego što počne izvršenje aplikacionih programa. Inicijalizaciona rutina koja je obično smeštena u ROM postavlja sve konstante, adrese i druge podatke na stanje koje odgovara početku rada sistema. Inicijalizacijom se može garantovati jedna tipična memorijska mapa kao što je ona prikazana na slici 4.9.

Inicijalizaciona procedura se deli na dve faze. U prvoj fazi, resetovan je hardver CPU-a, a to uslovljava izvršenje rutine sa lokacije određene početnom vrednošću PC-a. Ovaj deo inicijalizacije određen je od strane projektanata procesora i ne može se menjati. Druga faza počinje sa inicijalizacionom rutinom koju izvršava supervizorski program.



Sl. 4.9. Primer memorijske mape.

Reset i inicijalizacija procesora - aktivira se $\overline{\text{RESET}}$ linijom. Sekvenca događaja je prikazana na slici 4.10.

Posle reseta procesor čita 32-bitni podatak sa lokacija (00-03) i smešta ga u supervizorski magacin, a nakon toga 32-bitnu vrednost sa lokacija (04-07) smešta u PC. Hardver sistema treba da obezbedi da se ovih osam bajtova čitaju kao permanentne vrednosti.

Vektorska tabela izuzetaka

U Tabeli 4.5 prikazana je lista svih vektor lokacija koje se odnose na izuzetke. Tabelu čine 256 32-bitnih adresa od kojih svaka pokazuje na lokaciju odgovarajuće rutine za obradu izuzetaka, tj pridružena je tom izuzetku. Samo prve dve lokacije koje ukazuju na reset adresu su obavezne. Ostale lokacije se loaduju od strane inicijalizacione rutine u saglasnosti sa zahtevima definisanim od strane projektanta hardvera i softvera sistema. Obično da bi se izbegle nepoželjne akcije sistema, kada se javi neočekivani izuzetak, neiskorišćenim vektorima izuzetka u vektor tabeli se dodeljuje zajednička adresa koja ukazuje na početak neke opšte rutine za obradu izuzetka.

Nakon inicijalizacije, vektorska tabela zauzima prvih 1024 memorijskih lokacija. U suštini sadržaj VBR-a (Vector Base Register) je $0000 0000 i određuje početnu adresu vektorske tabele. Proizvoljna vektor adresa određuje se od strane CPU-a na sledeći način:

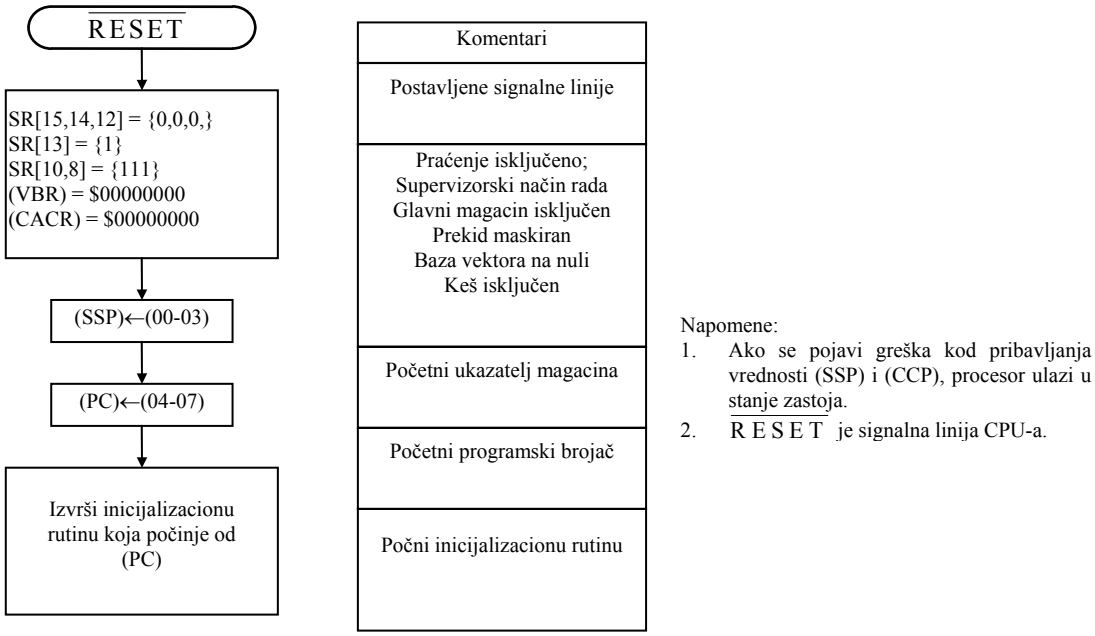
vektorska adresa=(VBR)+<ofset>

gde je vrednost ofseta definisana Tabelom 4.5. Ako OS rutina promeni sadržaj VBR-a koristeći instrukciju MOVEC, vektorska tabela se može premestiti u memoriju.

CPU automatski predaje upravljanje rutini za obradu izuzetka definisanu od strane odgovarajućeg vektora u trenutku kada se javi izuzetak. Zbog toga, odziv na bilo kakv izuzetak koji je različit od izuzetka reset nije određen od strane CPU-a nego u potpunosti zavisi od aktivnosti koje su predviđene od strane te rutine. Rutine za obradu izuzetaka se mogu podeliti na dve kategorije: one koje pripadaju monitor programu ili OS-u, i one koje su kreirane od strane programera sistema.

Rutina za inicijalizaciju - ova rutina priprema sistem za izvršenje supervizorskog programa. Ovu pripremu čine: punjenje vektorskih adresa na lokacije iz Tabele 4.5, i inicijalizacija spoljnih uređaja. Kada rutina završi, upravljanje se predaje supervizorskoj rutini, koja određuje koji će se aplikacioni program izvršavati. Ako ne postoji supervizor, upravljanje nakon završetka inicijalizacije se predaje direktno aplikacionom programu.

Rutina za inicijalizaciju koja se odnosi na sistem zasnovan na mikroprocesoru MC68020 obično obavlja niz operacija definisanih u Tabeli 4.6. U zavisnosti od aplikacije, varijacije izvođenja ove rutine su moguće.



Sl. 4.10. Sekvenca događaja kod reseta i inicijalizacije procesora.

Stanja: Normalno, Izuzetak i Zastoj

Kao što smo već uočili na slic 4.8 mikroprocesor može da bude u jednom od sledećih procesnih stanja: a) normalno, b) izuzetno i c) zastoj. U Tabeli 4.7 dat je sumarni pregled sva tri stanja CPU-a. Normalno stanje je karakteristično za izvršenje programa bilo u supervizorskom, bilo u korisničkom načinu rada. U ovom stanju CPU pribavlja instrukcije i operande, u toku izvršenja programa, iz memorije. U ovo stanje se ulazi nakon inicijalizacije sistema.

Specijalna situacija u toku normalnog stanja javlja se kada se ispuni uslov za zaustavljanje. U ovom slučaju, CPU na dalje ne izvršava instrukcije, nego čeka na neki spoljni događaj koj će inicirati da izvršenje produži.

Uslov zaustavljanja u normalnom stanju se javlja kada se izvrši instrukcija

STOP #<d16>

16-bitna neposredna vrednost <d16> zamenjuje sadržaj SR-a, a (PC) ukazuje na narednu naredbu. Sve dok se prekid ili drugi događaj ne javi, CPU zaustavlja pribavljanje i izvršavanje naredbi. STOP instrukciju mora da izvrši program u supervizorskom načinu rada, ili da izvrši neki privilegovani prekršaj (trap). U praksi, instrukcija STOP se koristi kao instrukcija "čekaj prekid".

Tab. 4.5. Vektorska tabela izuzetaka za MC68020.

Vektorski brojevi	Vektorski ofset		Dodela
	Heksa	Prostor	
0	000	SP	Reset: Inincijalni ukazatelj magacina prekida
1	004	SP	Reset: Inicijalni programski brojač
2	008	SD	Greška na magistrali
3	00C	SD	Adresna greška
4	010	SD	Ilegalna instrukcija
5	014	SD	Deljenje nulom
6	018	SD	Instrukcije CHK, CHK2
7	01C	SD	Inatrukcije cpTRAPcc, TRAPcc i TRAPV
8	020	SD	Narušavanje privilegije
9	024	SD	Režim praćenja
10	028	SD	Emulator linije 1010
11	02C	SD	Emulator linije 1111
12	030	SD	(Nedodeljeno, rezervisano)
13	034	SD	Narušavanje koprocesorskog protokola
14	038	SD	Greška formata
15	03C	SD	Neinicijalizovani prekid
16 do 23	040 do 05c	SD	(Nedodeljeno, rezervisano)
24	060	SD	Lažni prekid
25	064	SD	Auto vektor prekida nivoa 1
26	068	SD	Auto vektor prekida nivoa 2
27	08C	SD	Auto vektor prekida nivoa 3
28	070	SD	Auto vektor prekida nivoa 4
29	074	SD	Auto vektor prekida nivoa 5
30	078	SD	Auto vektor prekida nivoa 6
31	07C	SD	Auto vektor prekida nivoa 7
32 do 47	080 do 08C	SD CD	(Nedodeljeno, rezervisano)
48	0C0	SD	FPCP grananje ili postavi na neuređeni uslov
49	0C4	SD	FPCP netačan rezultat
50	DC8	SD	FPCP deljenje nulom
51	0CC	SD	FPCP potkoračenje
52	0D0	SD	FPCP greška operanda
53	0D4	SD	FPCP prekoračenja
54	0D8	SD	FPCP signaliziranje NAN
55	0DC	SD	Nedodeljeno, rezervisano
56	0E0	SD	PMMU konfiguracija
57	0E4	SD	PMMU ilegalna operacija
58	0E8	SD	PMMU narušavanje nivoa pristupa
59 do 63	0EC do 0FC	SD	(Nedodeljeno, rezervisano)
64 do 266	100 do 3FC	SD	(Nedodeljeno, rezervisano)

SP = Supervizorska programska oblast
SD = Supervizorska oblast podataka

U stanje izuzetak se ulazi kada se javi: reset, prekid, trap ili neki drugi izuzetak. CPU automatski prelazi u supervizorski način rada i obrada izuzetka počinje. Kao što je prikazano na slici 4.11 sekvenca izuzetka se deli na dva dela. Obrada izuzetka, za najveći broj izuzetaka, uključuje sledeće aktivnosti: promenu načina rada CPU-a u supervizorski, određivanje vektor broja u CPU-ovoj vektorskoj tabeli koji prati taj izuzetak, pamćenje stanja SR-a, PC-a i drugih informacija u supervizorski magacin. Nakon toga vektor adresa koja ukazuje na rutinu za obradu izuzetka se automatski puni u PC i upravljanje se predaje toj rutini. CPU, nakon toga, sve dok rutina ne završi, prelazi u normalnu obradu. Na kraju rutine, naredbom RTE, obnavlja se staro stanje CPU-a, tj. izbavljaju se vrednosti za (SR), (PC) i druge informacije iz supervizorskog magacina i smeštaju u registre CPU-a.

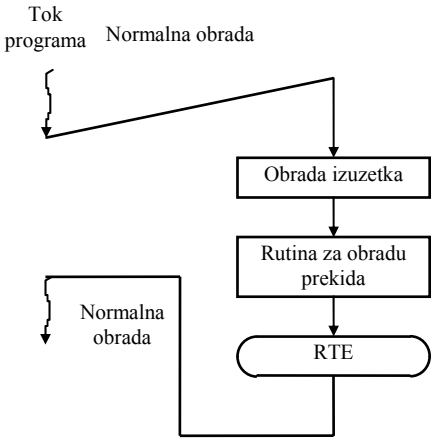
Tab. 4.6. Tipična procedura inicijalizacije sistema.

Operacija	Komentari
Inicijalizacija zahtevane vektorske adrese.	Lokacije $0008-$03FC (32-bitna adresa)
Napuni potrebne supervizorske rutine.	Napuni sa diska jedinice
Inicijalizuj sadržaje svih memorijskih lokacija koje su potrebne supervizoru.	koje su potrebne
Inicijalizuj sve periferalne servise sistema.	Kao što je potrebno: RESET instrukcija za periferalne uređaje i druge U/I upravljačke komande
Inicijalizuj USP.	(USP) ← adresa korisničkog magacina
Upiši u magacin reč formata.	((SSP)) ← reč formata
Upiši u magacin početnu adresu za program u korisničkom načinu rada.	((SSP)) ← početak programa
Smesti u magacin stanje programa u korisničkom načinu rada.	((SSP)) ← korisnički (SR)
Omogući keš.	(CACR) = $00000001
Prebaci upravljanje programu u korisničkom načinu rada.	RTE

Tab. 4.7. Stanja i načini rada procesora.

Stanje	Uslov	CPU aktivnost
Normalno	Obrada	Izvršavanje programa u supervizorskom ili korisničkom načinu rada
	Zaustavljeno	Čekanje na prekid
Izuzetak	Reset	Inicijalizacija
	Prekid	Prihvatanje i obrada prekida
	Trap	Obrada trapa
	Praćenje	Izvršavanje jedne po jedne instrukcije ili praćenje promene toka
Zastoj	Uslov greške sistema	

Napomena: Izuzeci pobrojani u ovoj tabeli su izabrani primenri.



Napomena: Obrada izuzetka se automatski izvršava od strane CPU kada se prepozna izuzetak. Upravljanje se tada prenosi rutini za obradu izuzetaka koja je pridružena određenom izuzetku koji se pojavio.

Sl. 4.11. Sekvenca izuzetka.

Stanje zastoj (*halted*) obezbeđuje zaštitu sistema na taj način što se uslovljava da CPU prekine sve spoljne aktivnosti. Procesor se zaustavlja ako detektuje određeni tip greške, dok je u toku obrada drugog tipa greške. Ovakve greške se mogu javiti samo nakon katastrofalnih hardverskih grešaka čije otkrivanje nije moguće. U tom slučaju je potrebno resetovati sistem da bi ponovo startovali zaustavljeni procesor. U toku stanja zastoj, CPU generiše informaciju o svom stanju preko specijalnih signalnih linija.

4.4.3. Sinhronizacija procesa

RS čini veći broj procesa. Procesi se mogu izvršavati paralelno (simultano) ako RS čini veći broj procesa. Ako postoji samo jedan procesor, paralelni rad se može simulirati ako procesor izvršava (delimično) svaki proces za kratki vremenski interval a zatim prelazi na izvršenje drugog procesa, tj. procesor se deli između nekoliko procesa. Probleme koji se javljaju u toku izvršenja paralelnih procesa ilustrovaćemo na sledećem primeru. Analizirajmo dva procesa:

```
process Observer
   repeat
        Observe_an_event;
        Count:=Count+1;
   until false;

process Reporter
   repeat
        Print_count;
        Count:=0;
   until false;
```

Prvi proces, Observer, odgovoran je za nadgledanje i brojanje događaja. Drugi proces Reporter, povremeno štampa izveštaje o broju događaja, a zatim postavlja brojač na nulu. Kada se oba procesa izvršavaju konkurentno, može da se javi sledeća situacija. Pretpostavimo da je Reporter upravo odštampao 10, i pre nego što postavi brojač na nulu, Observer poveća brojač na 11 (na primer, kada se izvršenje Reportera prekine, a Observera planira za izvršenje kao naredno). Nakon toga, Reporter se planira za izvršenje ponovno i briše brojač, što znači da zadnji događaj ostaje neevidentiran. U opštem slučaju, Reporter može pogrešno dati podatak o broju događaja jer se inkrementiranje promenljive Count može desiti između štampanja vrednosti Count i njenog postavljanja na nulu. Ovaj primer ukazuje da su rezultati nepredvidljivi, jer se procesi izvršavaju sa nepredvidljivim relativnim brzinama. Kada rezultat izražunavanja zavisi od relativnih brzina procesa kažemo da se javlja trka uslova. Trka uslova javlja se kada paralelni procesi dele podatke ili resurse.

Problem trka uslova rešava se ako se uvede korektno upravljanje redosledom po kome procesi operišu nad deljivim podacima ili resursima. Ovo se zove sinhronizacija procesa. Glavni sinhronizacioni problem je problem uzajamne isključivosti. Uzajamna isključivost garantuje da će najviše jedan proces pristupati deljivim podacima ili resursima. Procesi koji pristupaju deljivim podacima ili resursima tipično to izvode preko malih sekcija ili programa koje se zovu kritične sekcije.

Kada se izvrši pokušaj da se reši problem uzajamne isključivosti, moramo biti svesni situacije smrtnog zagrljaja. Smrtni zagrljaji se javljaju kada procesi čekaju na događaj koji se nikad neće javiti. Na primer, smrtni zagrljaj se može javiti u sledećoj situaciji. Neka postoje dva procesa P1 i P2 i dva resursa R1 i R2. Dalje usvajamo da oba procesa imaju potrebu za oba resursa istovremeno kako bi završili. Ako se R1 dodeli P1, pre nego što P1 zahteva R2, a R2 se dodeli procesu P2, javiće se smrtni zagrljaj, jer će se oba procesa blokirati. Ovo ukazuje da i pored toga što se uzajamnom isključivošću izbegava problem trke uslova, to nije dovoljno da imamo konkurentne procese koji kooperišu korektno i efikasno, koristeći deljive podatke. Sledeća četiri uslova treba da važe da bi se dobilo korektno rešenje:

1) U bilo kom trenutku, može biti najviše jedan proces u zadatoj kritičnoj sekciji. To je zahtev za uzajamnom isključivošću.
2) Ne smeju se uvoditi pretpostavke o relativnim brzinama procesa, sa ciljem da se izbegnu trke uslova.
3) Proces koji je stopiran van kritične sekcije ne bi trebali da blokira bilo koji drugi proces, čime se izbegava smrtni zagrljaj.
4) Procesi koji su spremni za ulazak u kritičnu sekciju ne treba neograničeno da blokiraju jedan drugog, da bi se izbegao problem "umiranja-od-gladi". Umiranje-od-gladi znači da se određeni proces neće nikad izvršiti, jer drugi procesi kontinualno zauzimaju njegov red na jedan nekorektan način.

Veći broj sinhronizacionih metoda, na hardverskom, firmverskom, kao i softverskom nivou se koriste da bi se rešio problem uzajamne isključivosti.

Zabrana rada prekidima

Najjednostavnije rešenje problema uzajamne isključivosti je kada proces zabrani rad svim prekidima nakon ulaska u kritičnu sekciju, a ponovo dozvoli njihov rad kada je napusti. Na ovaj način, operacije koje pristupaju deljivim promenljivim, ili resursima, se čine nevidljivim ili atomskim. Operacija je nevidljiva ili atomska ako se njeno izvršenje ne može prekinuti. Prednost rešavanja problema uzajamne isključivosti zabranom rada prekida ogleda se u njenoj jednostavnosti, ali ne zaboravimo da se javljaju i teškoće. Prvo, kritične sekcije moraju biti

kratke, jer se prekidi neće na vreme prihvatiti, a U/I uređaji neće, takođe, na vreme biti opsluženi. Kao drugo, ovaj metod je jedino pogodan za jednoprocesorske sisteme, jer kod multiprocesorskih sistema zabrana rada prekida na jednom procesoru ne zaustavi procese na drugim procesorima, tako da oni mogu da uđu u kritičnu sekciju. Jedan od načina da se izbegne ovaj problem je da se projektuje multiprocesorski sistem tako da samo jedan od procesa prihvata i obrađuje sve prekide.

Zauzetost čekanjem na signal (Busy waiting)

Kao alternativa zabrani prekida, za upravljanje pristupom kritičnoj sekciji, koristi se Lock promenljiva. Promenljivom Lock upravlja se pristupom kritičnoj sekciji na sledeći način: ako je Lock=0 može se ući u kritičnu sekciju; ako je Lock=1 kritična sekcija je blokirana.

```
Loop :  TST       Lock     ; Napomena: ako je Lock=0 kritična sekcija je slobodna
        BNE       Loop     ; Kritična sekcija je blokrana
        ADDQ.B    #1,Lock  ; Zauzmi kritičnu sekciju
```

Izvršenje programa može biti prekinuto - na primer, nakon izvršenja TST instrukcije. Rezultat će biti nepredvidljiv ako procesu koji je izvršio prekid bude dozvoljeno da uđe u kritičnu oblast. Da bi se zaštitili od ovoga, program treba da se izvršava tako što će se prekidi zabraniti kao što je prikazano na sledećem primeru.

```
Loop :  Zabrani prekide
        TST       Lock
        BEG       Free     ; Može se ući u kritičnu sekciju
        Dozvoli prekide
        BRA       Loop     ; Kritična sekcija je blokirana
Free :  ADDQ.B    #1,Lock  ; Zauzmi kritičnu sekciju
        Dozvoli prekide
```

Veliki broj računara ima specijalne instrukcije koje podržavaju atomske akcije. Najpoznatije su TAS, CAS i lock meta-instrukcija.

TAS instrukcija

Kod MC68020, instrukcija TAS je nedeljiva i realizuje se kao memorijski ciklus čitaj-modifikuj-upiši. U toku ovog memorijskog ciklusa put memoriji drugim procesorima je blokiran (to mogu biti drugi CPU-ovi kod multiprocesorskih sistema, DMA kanali ili U/I procesori).

Sledeća programska sekvenca prikazuje način korišćenja instrukcije TAS.

```
Wait :  TAS       Lock     ; Ispitaj da li je kritična sekcija slobodna
        BMI       Wait     ; Čekaj jer je LockŠ7Ć=1
        Kritična sekcija   ; Izvrši operaciju koja sledi u nizu
        CLR.B     Lock     ; Oslobodi kritičnu sekciju
```

Proces repetitivno testira promenljivu Lock, koristeći instrukciju TAS, sve dok ne detektuje da je MS bit Lock promenljive resetovan. TAS postavlja MS bit kao rezultat nedeljive operacije, nakon toga proces ulazi u kritičnu sekciju a briše promenljivu Lock kada je napušta. Ova tehnika zove se "busy waiting" ili "spin-lock".

Instrukcija CAS

TAS instrukcija upravlja se pristupom kritičnoj sekciji na osnovu logičke promenljive Lock promenljive, što čini da ova instrukcija bude veoma jednostavna. Nasuprot tome CAS instrukcijom upravlja se pristupom kritičnoj sekciji preko celobrojne Lock promenljive. CAS instrukcija se čini nedeljivom zabranom prekida i korišćenjem memorijskog ciklusa čitanje-modifikacija-upis. CAS instrukcija se može koristiti za rešavanje problema uzajamne isključivosti na sledeći način

```
        MOVEQ     #1,D2          ; Postavi blokirajuću vrednost i
                                 ; ažuriraj vrednost u D2
Loop:   CLR.W     D1             ; Obriši operand za upoređivanje u D1
        CAS.W     D1,D2,Lock
        BNE       Loop
        Kritična sekcija
        CLR.W     Lock           ; Oslobodi kritičnu sekciju
```

Ako je Lock=0, što je vrednost u kompariranom operandu D1, sadržaj ažuriranog operanda u D2 se smešta u Lock, što ukazuje da je proces ušao u svoju kritičnu sekciju. Inače, ako je Lock=1 proces se vraća na Loop i pokušava se iznova ulazak u kritičnu sekciju.

Primer 4.1:

Sledeća programska sekvenca prikazuje deo procesa koji dodaje vrednost *N* vrednosti brojača koja je smeštena u D1. Brojač se može ažurirati od strane nekoliko procesa. Program se može izvršavati od strane proizvoljnog broja konkurentnih procesa.

```
        MOVE.W    Count,D0      ; Iskopiraj tekuću vrednost Count u D0
Loop :  MOVE.W    D0,D1         ; Napravi kopiju vrednosti Count
        ADD.W     #N,D1         ; Uvećaj Count za N
        CAS.W     D0,D1,Count   ; Ako neki drugi proces nije promenio vrednost Count
                                ; u međuvremenu, ažuriraj je
        BNE       Loop          ; Ako je vrednost Count promenjena od strane drugog
                                ; procesa, pokusaj ponovo
```

Neke arhitekture kao što je MC68020, imaju i CAS2 instrukciju koja je slična po načinu rada kao i CAS, ali ima veći broj operanada. CAS2 omogućava da se manipulacija sa redivima čekanja izvodi automatski.

LOCK meta-instrukcija

Kod mikroprocesora 80x86 iz familije Intel, postoji specijalna jednobajtna LOCK prefiks instrukcija. Ona se može koristiti sa bilo kojom drugom instrukcijom, koja se tada zove "locked instrukcija", ako se postavi ispred te instrukcije. Ovaj prefiks čini locked instrukciju nedeljivom. LOCK se može koristiti u narednoj programskoj sekvenci. Instrukcija XCHG (kod iAPX 286, iAPX 386 i iAPX 486) implicitno uzrokuje zauzimanje magistrale, pa ne postoji potreba za LOCK prefiksom kod test-and-set operacije.

```
Loop :  MOV    AL,1                ; Kopiraj 1 u registar opšte namene AL
        LOCK   XCHG AL,Lockvar     ; Izmeni Al sa lock promenljivom, sada će kritična
                                   ; sekcija biti blokirana zbog Lockvar = 1
        CMP    AL,0                ; Testiraj da li je kritična sekcija bila na početku
                                   ; slobodna. tj. Lockvar = 0
        JNE    Loop                ; Kritična sekcija blokirana, pokusaj ponovo
        Kritična sekcija
        MOV    Lockvar,0           ; Oslobodi kritičnu sekciju
```

Tab. 4.8. Sinhronizacione konstrukcije i implementacione primitive.

Koncepti	Implementacione primitive
Monitori	Mehanizam modula
Događaji	Čekaj, izazovi
Semafori	P i V
Lock promenljive	TAS, CAS, LOCK metainstrukcije
Zabranjivanje	Prekid dozvoljen/zabranjen

Konstrukcije za sinhronizaciju višeg nivoa

Doskora, uzajamna isključivost je bila implementirana korišćenjem tehnike "busy waiting". Ova tehnika, na žalost, dovodi do gubitaka CPU-ovog vremena, čak i kada je svakom procesu dodeljen po jedan CPU, što čini da busy-waiting tehnika bude nepraktična. Problem je što busy-waiting može da zasiti put ka memoriji, što može da uspori U/I aktivnosti i druge CPU-ove (kod multiprocesorskih sistema). Dijkstra (1956) je uveo semafore, čime se problem "busy waiting" izbegava suspenzijom (smeštanjem blokiranih procesa u stanje čekanja) a zatim postavljanjem procesa u stanje -spremni- kada se napusti kritična oblast. Semafori se mogu implementirati koristeći TAS i/ili CAS instrukcije. Druge konstrukcije za sinhronizaciju, višeg nivoa, kao što su događaji (events) i monitori se mogu takođe implementirati TAS i/ili CAS instrukcijama. Ove konstrukcije poseduju veći nivo apstrakcije (Tabela 4.8).

4.4.4. Komutacija procesa

Kako svi procesi treba da dele istu CPU, CPU mora da vrši komutaciju procesa. Komutaciju procesa treba sagledati kroz:

a) prizivanje komutacije procesa - ukazuje zašto se i kada javlja komutacija procesa,
b) komutaciju konteksta - ukazuje koji kontekst treba sačuvati i kako.

Prizivanje komutacije procesa

Komutaciju procea je moguće prizvati na nekoliko načina. Na slici 4.12. označena je komutacija između procesa. Obrnuti smer komutacije zbog jednostavnijeg označavanja na slici 4.12 nije naznačen. Komutacija procesa se može izvesti na račun:

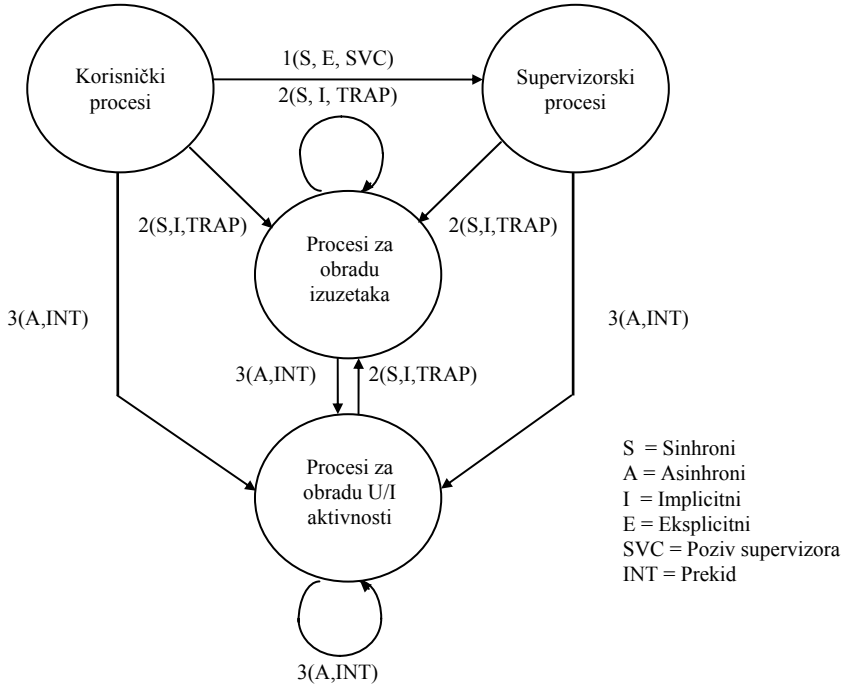
i) procesa koji se trenutno izvršava,
ii) drugog procesa.

Komutacija procesa na račun procesa koji se trenutno izvršava

Javlja se kada je tekućem procesu, koji se nalazi u stanju "running", potrebna usluga od strane procesa na nižem nivou. Komutacija se izvodi sinhrono što znači da za proces koji je prozvan možemo da smatramo da će se izvršiti serijski sa procesom koji je izvršio poziv. Poziv se može izvesti na dva načina:

- eksplicitnim pozivanjem,
- implicitnim pozivanjem.

Eksplicitno pozivanje - zove se poziv supervizora (supervisor call - SVC), predstavlja procesnu komutaciju sa korisničkog na supervizorski proces (1 na slici 4.12). SVC se koristi od strane korisničkog procesa kada on zahteva da se obavi OS funkcija, ili da naznači da je on spreman (završio je). Najveći broj arhitektura poseduje specijalnu instrukciju pomoću koje se ostvaruje poziv supervizora (potprogramski pozivi ili pozivi procedura se ne mogu koristiti, jer one dozvoljavaju da se kao ulazna tačka rutine koja se poziva bude specificirana bilo koja memorijska lokacija, što dovodi do nesigurnosti u radu). Prednost specijalne "supervisor call" instrukcije je u tome što ona može da sadrži parametar koji ukazuje na to koju OS uslugu je potrebno pozvati.



Sl. 4.12. Komutacija procesa.

Primer 4.2:

MC68020 podržava poziv supervizora preko TRAP instrukcije. TRAP instrukcija (slika 4.13.) ima četvorobitni operand koji se koristi za specifikaciju vektor broja. Instrukcijom se prvo smešta u magacin vektor broj, PC i SR, a zatim se PC puni na početnu adresu supervizorske rutine koja se poziva. Ova početna adresa se određuje "vector" poljem instrukcije TRAP, koja omogućava da se ostvari 16 različitih supervizorskih poziva.


```
TRAP      #<Vector>
SSP:=(SSP) - 2        ; Dekrementiraj ukazatelj magacina
M[(SSP)]:=Vector      ; Sačuvaj broj vektora
SSP:=(SSP) - 4
M[(SSP)]:=(PC)        ; Sačuvaj PC
SSP:=(SSP) - 2
M[(SSP)]:=(SR)        ; Sačuvaj SR
PC:=(M[Vector])       ; Napuni PC početnom adresom
```

Sl. 4.13. TRAP instrukcija.

Implicitno pozivanje - zove se trap ili interni prekid (2 na slici 4.12), i predstavlja komutaciju na proces za obradu izuzetka. Poziva se automatski od strane hardvera računara kada se detektuje neobičan ili izuzetan uslov. Primeri ovakvih uslova su greške u podacima (kao što su premašaj, podbačaj ili deljenje nulom), nekorektnost u toku rada koja se odnosi na memorisanje podataka, nedefinisane ili privilegovane opkodove i premašaj magacina. Za interne prekide se smatra da su sinhrone komutacije procesa, jer oni vrše prenos upravljanja procesu za obradu izuzetka koji se izvršava na račun (i sekvencijalno sa) procesa (procesom) koji vrši prizivanje. Kod MC68020, interni prekidi se generišu od strane:

a) DIVU i DIVS instrukcija ako je delitelj nula,
b) instrukcije CHK kada je operand negativan ili veći od gornje specificirane granice,
c) TRAP instrukcije koja uslovljava procesnu komutaciju samo kada je bit prekoračenja (V) postavljen.

Komutacija procesa na račun drugog procesa

Dok hardver CPU-a izvršava proces, drugi procesi (koji koriste različiti hardver) mogu takođe biti aktivni. Ovi procesi mogu biti U/I procesi (disk ili terminalni U/I), ili proces satnog mehanizma. Kada ovi drugi procesi zahtevaju opsluživanje od strane procesa koji se izvršava na CPU-u, oni signaliziraju da zahtevaju pažnju preko signalne linije za spoljni prekid. Kada se ovaj zahtev prizna, spoljni prekid se prihvata, a U/I rutina za obradu procesa se poziva (3 na slici 4.12). Ovo se izvodi asinhrono sa izvršenjem procesa čije se izvršenje prekida, jer spoljni prekid nije bio zatražen od strane tog procesa. Komutacija procesa može, zbog toga, da se javi u bilo kom vremenskom trenutku i nije u nikakvoj uzajamnoj vezi sa procesom koji se trenutno izvršava.

Opšti zaključak bi bio sledeći. Uzrok komutacije procesa može biti: poziv supervizora, interni prekid (trap) i spoljni prekid (često se zove kratko - prekid). CPU koristi jedinstveni hardverski mehanizam - mehanizam prekida, kojim se vrši pozivanje sve tri klase komutacije procesa, a sve se one mogu označiti kao prekidi. O načinu rada mehanizama prekida govorićemo kasnije.

Komutacija konteksta

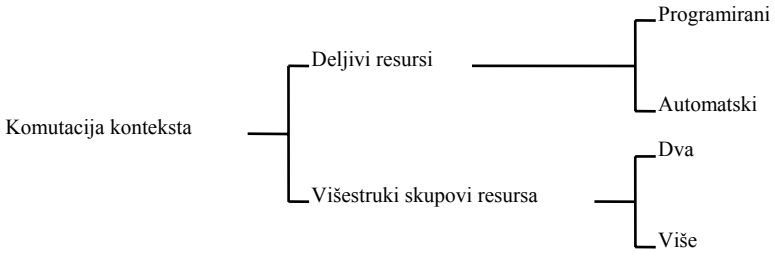
Kod svake komutacije procesa, status procesa koji se trenutno izvršava mora da se sačuva, novi proces izabere, a kontekst tog procesa mora da se obnovi. Komutacija konteksta mora biti brza, jer vreme koje se gubi zbog nje ne sme da ima negativan efekat na korisnički program. Šta više, spoljni uređaji često zahtevaju brz odziv na njihov zahtev za prekid, tako da se kao imperativ nameće zahtev da se u toku komutacije konteksta gubi što je moguće manje vremena.

Komutacija konteksta, kao što je prikazano na slici 4.14, se može izvesti na veći broj načina. Ako je dostupan samo jedan skup hardverskih resursa (na primer, registri procesora) oni se moraju deliti između različitih konteksta. Nakon komutacije konteksta, tekuća vrednost koja je memorisana u ovim deljivim resursima, mora da se sačuva, tako da se oni mogu napuniti vrednostima novog procesa. Kod povratka na početni proces, stanje ovih resursa se mora obnoviti na stare vrednosti. Memorisanje i obnavljanje se izvodi automatski kao deo operacije komutiranja konteksta, ili pod programskim upravljanjem.

Alternativa deljivim resursima je obezbeđenje većeg broja skupova resursa, pri čemu svaki skup sadrži kontekst. Ovo omogućava da se komutacija konteksta svodi na izbor različitog skupa resursa.

Komutacija na različite delove konteksta, kao što je prikazano na slici. 4.3, čini komutaciju konteksta koja:

- se obavlja od strane operacije prekida,
- je deo komutacije konteksta prostora magacina i memorije,
- je deo komutacije konteksta ostatka konteksta procesora.

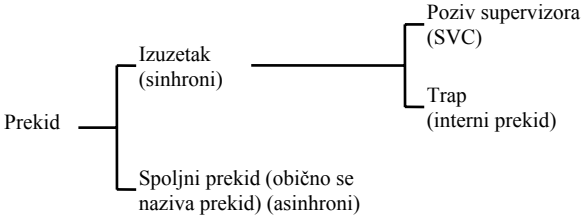


Sl. 4.14. Načini izvođenja komutacije konteksta.

Komutacija konteksta koja se obavlja operacijom prekida

Komutacija između procesa se inicira prekidom. Na slici 4.15 prikazana je klasifikacija prekida.

Prekid koji poziva proces može se uporediti sa pozivima potprograma i procedura. Razlika je u iznosu procesorskog konteksta. Procedure se izvršavaju u okruženju procesa; njihov kontekst je podskup onog koji pripada procesu. Procesorski kontekst procedure čine podaci i adresni registri CPU-a (a takođe i koprocesora) zajedno sa internim stanjem CPU-a. Procesorski kontekst procesa dodatno sadrži PSW (specificira izbor privilegovanog skupa naredbi i nivo prioriteta procesa) kao i informaciju koja se odnosi na upravljanje memorijom (smeštena je u upravljačkim registrima), kojom se specificira kontekst memorije i pristup informaciji od strane procesa. Sve ovo čini ukupnu informaciju koju treba sačuvati/izbaviti nakon komutacije procesa.



Sl. 4.15. Klasifikacija prekida.

Prekidi, isti se zaključak odnosi na poziv potprograma i procedura, ne moraju da memorišu/obnavljaju celokupni kontekst procesora jer to čini da ove operacije budu mnogo kompleksne i da dugo traju. Određeni delovi kontekst aktivnosti se mogu lako izvoditi koristeći instrukcije tipa MOVEM.

Minimalni procesorski kontekst koga treba memorisati/izbavljati od strane mehanizma za prekid je onaj koji obezbeđuje da prizivani proces završi procesnu kontekst komutaciju. Mehanizam prekida treba, zbog toga, da memoriše/izbavi sadržaj sledećih resursa u toku jedne nedeljive operacije:

- statusnog registra (jer on je jedinstveni deljivi resurs)
- PC-a (jer je jedinstveno deljivi resurs)
- pokazivača magacina (jer svaki proces ima svoj sopstveni magacin)
- informacije za upravljanje memorijom (da bi se komutirao kontekst memorije).

Pošto minimalni procesorski kontekst mora da se komutira kao automatska akcija, deljivi registri kao što su SR i PC moraju se automatski memorisati, ili kada se koristi veći broj registarskih skupova (kao kod mikroprocesora T9000 iz serije Texas Instruments), izbor različitog (novog) skupa mora da se izvede automatski.

Obrada prekida, a to znači prizivanje novog procesa se obavlja na sledeći način:

- memoriše se minimalni procesorski kontekst tekućeg procesa,
- puni se minimalni kontekst procesora novog procesa. Vrednost SR-a se tipično generiše od strane hardvera za prekid. Vrednost za PC se dobija iz fiksne lokacije u memoriji, određene od strane prekidnog vektora, koji specificira proces koji se poziva. Kada proces završi, on koristi instrukciju za povratak, koja poziva prekinuti proces. Ova privilegovana instrukcija obnavlja vrednosti SR-a i PC-a.

Komutacija konteksta memorijskog prostora i prostora magacina

Kao što se vidi sa slici 4.12, postoje četiri klase procesa. Sa tačke gledišta zaštite memorije oni se mogu podeliti u dve grupe: a) korisnički procesi, b) sistemski procesi (formirani od strane supervizora, U/I i procesi za obradu izuzetaka). Korisničke procese čine korisnički programi koji mogu da sadrže greške ili programe na osnovu kojih se može narušiti mehanizam zaštite. Sistemski procesi, sa druge strane, se mogu smatrati pouzdanim i

testiranim. Zbog toga kada se kaže upravljanje memorijom obično se misli na rad sa dva memorijska prostora: jedan je namenjen za korisnički proces, a drugi za sistemske procese. Memorijski prostor sistemskih procesa se dalje može deliti na podprostore za svaki od sistemskih procesa.

U zavisnosti od informacije o komutaciji konteksta, koja se odnosi na upravljanje memorijom, moguće je da se jave sledeće situacije:

- Komutacija procesa sa korisničkog na sistemski proces: Kako i korisnički i sistemski procesi, svaki za sebe, imaju svoje sopstvene memorijske prostore, ovi prostori moraju da se komutiraju. Da bi ubrzali ovu operaciju, najveći broj sistema za upravljanje memorijom su tako projektovani da imaju dva skupa registara, u kojima se čuva kontekst svakog prostora. Ovo se izvodi pod kontrolom bita "Supervisor/User" koji pripada registru SR.
- Komutacija procesa sa sistemskog na korisnički proces: U ovom slučaju se mogu desiti dve situacije. Prva se odnosi na povratak sa sistemskog procesa na korisnički proces koji je izvršio poziv sistemskog procesa. U ovoj situaciji se zahteva da samo kontekst koji se odnosi na upravljanje memorijom korisničkog prostora bude ponovo selektovan pod kontrolom bita "Supervisor/User" koji pripada registru. Druga situacija se javlja kada sistemski proces komutira na neki drugi korisnički proces (zbog U/I čekanja ili isteka "time-out"-a). Da bi se obavila komutacija, poziva se rutina kojom se vrši memorisanje konteksta upravljanja memorijom korisničkog procesa koji se trenutno izvršava, i obnavlja se stanje korisničkog procesa na koga se vrši komutiranje. Nakon toga, bit "Supervisor/User" se automatski postavlja na stanje koje odgovara korisničkom načinu rada.
- Komutacija procesa između sistemskih procesa: Pošto svi sistemski procesi imaju isti kontekst upravljačke memorije, nema potreba za preduzimanjem specijalnih akcija.

Prostor magacina je deo prostora namenjen procesu. Ako postoje različiti prostori magacina za korisničke i sistemske procese, komutacija procesa uključuje takođe i komutaciju prostora magacina. Obično se ova aktivnost sastoji u izboru odgovarajućeg pokazivača magacina.

Komutacija konteksta ostatka procesorskog konteksta

Minimalni procesorski kontekst je već memorisan kao deo operacije prekida. Ostatak procesorskog konteksta (na primer, registri opšte namene, i registri numeričkog koprocesora koji se mogu takođe koristiti od strane drugih procesa) mora se takođe komutirati. Kada su određeni registri namenjeni nekom procesu a ne koriste se od strane drugih procesa, ne moraju se komutirati. Ova komutacija konteksta se može izvesti: a) automatski kao rezultat komutacije procesa pa se u tom slučju svi registri memorišu, ili, b) pod programskom kontrolom - komutaciju konteksta obavlja novoselektovani proces. U ovom slučaju, samo oni registri koji se koriste od strane tog procesa moraju da se memorišu koristeći, na primer, MOVEM instrukciju. Kontekst se obično pamti u magacin.

4.4.5. Obrada prekida

Kao što smo već ukazali procesi treba da komuniciraju. Potreba za komuniciranjem se ostvaruje preko zahteva za prekid: Prozivanje ove komunikacije predstavlja prekid. Kod RS-a u jednom trenutku, veći broj procesa može imati aktivirane zahteve za prekid, imajući u vidu da se veći broj procesa može istovremeno izvršavati paralelno. Izbor koji se odnosi na to koji će se prekid opslužiti zahteva uvođenje nekog vida šeme prioriteta, koja se zasniva na hijerarhiji procesa u sistemu. Proces koji se trenutno izvršava može da preda upravljanje drugom procesu u određenom vremenskom intervalu. Na primer, zbog zaštite kritične sekcije prekidi mogu biti zabranjeni. Prekidi mogu biti dozvoljeni u nekoliko tačaka u toku izvršenja instrukcije. Kada se javi prekid, neophodno je da se identifikuje izvor prekida (proces koji zahteva prekid).

Prioritet prekida

Već smo ranije uočili, na osnovu logičke strukture OS-a, da su procesi hijerarhijski uređeni. Prioritet obrade prekida zavisi od dva faktora: a) prioritet izvršavanja procesa (prioritet sa kojim će se pozvani proces izvršiti), b) prioritet prikazivanja procesa (prioritet sa kojim se operacija prekida javlja). Prioritet sa kojim procesi moraju da se izvršavaju zavisi od njihove urgentnosti, koja je određena njihovom funkcijom, na sledeći način:

- **Sigurnosni prekršaj i hardverski defekti** - obrađuju se od strane procesa za obradu izuzetka i imaju najviši prioritet. Samo kada se oni završe izvršenje drugog procesa može da startuje.
- **Kritična sinhronizacija u radu** - kada se poruke podataka primaju preko komunikacione linije, prekidi od strane komunikacionog interfejsa se generišu posle svakog primljenog bajta. Ove prekide treba opslužiti za kratak vremenski interval, kako se ne bi izgubila informacija.
- **Performanse** - pojedini korisnik iz određenih razloga može da ima visok nivo prioriteta.

Primer 4.3:

Procesi operativnog sistema UNIX System V koji se izvršavaju na sistemu zasnovanom na MC68020 imaju sledeće nivoe prekida:

Tab. 4.9.

Nivo	Proces
7	Obrada izuzetaka i satnog mehanizma (nemaskirajući)
6	Vremenski kritični U/I
5	Terminalni U/I procesi
4	U/I koji nisu vremenski kritični
3	Ne koristi se
2	Ne koristi se
1	Ne koristi se
0	Korisnički procesi

Kada se u toku izvršenja nekog procesa nižeg nivoa generiše zahtev za izvršenje procesa višeg nivoa, poželjno je da se ovaj zahtev prizna, jer je njegova urgentnost viša. Ovo znači da će se kod opsluživanja prekida javljati gnežđenje, koje je slično gnežđenju potprograma i poziva procedura.

Dozvola/zabrana rada prekida

Zbog neophodnosti da se prekidima: a) dodele osobine, b) zabrani rad; sve to u cilju da se ostvari sinhronizacija procesa, računari treba da su sposobni da odrede kada su prekidi dozvoljeni. Trapovi su uvek dozvoljeni i oni se ne mogu maskirati. Činjenica da je njihov prioritet prizivanja najviši garantuje da instrukcije koje uzrokuju trap i obavljaju prizivanje procesa pomoću koga se vrši obrada trapa moraju da formiraju jedinstvenu atomsku akciju. Drugi oblici prekida se mogu maskirati.

Veći broj periferala u svojim upravljačkim registrima imaju ugrađene markere za dozvolu/zabranu rada prekida IE (Interrupt Enable). Na osnovu stanja markera određuje se da li je ili ne prekid dozvoljen/zabranjen. Najjednostavniji RS za obradu prioritetno zasnovanih prekida ima dva režima rada u kojima se nalazi CPU, a to su:

a) prekidi su dozvoljeni - prekid se prihvata odmah nakon završetka instrukcije koja se trenutno izvršava,
b) prekidi su zabranjeni - zahtevi za prekid se ne prihvataju.

Fleksibilnija šema se dobija kada postoji nekoliko nivoa prioriteta koji su hijerarhijski organizovani. Kada procesor izvršava prekid na određenom prioritetu, tekući nivo prioriteta i niži nivoi su zabranjeni, dok su svi viši dozvoljeni.

Alternativna šema je ona pomoću koje se dozvoljava/brani rad svakog od pomenutih načina nezavisno. Kod ove šeme, specificirani bit oblik u Interrupt Mask Register (deo je CPU-a) određuje koji će se prekid prihvatiti. Svaki maskirajući bit ovog registra odgovara pojedinom uslovu za prekid ili grupi uslova, i samo oni uslovi koji imaju odgovarajuće bitove za maskiranje postavljene, su dozvoljeni. Svi ostali uslovi (koji odgovaraju nulama u IMR-u) su zabranjeni.

Trenutak prekidanja

Sinhroni prekidi se javljaju kao rezultat izvršenja tekuće instrukcije. Zbog toga, oni se uvek mogu obrađivati direktno. jedini problem je odrediti kakav će biti kraj te instrukcije za slučaj da se javi trap (trap se generiše zbog izuzetnog uslova). Postoje četiri alternative:

1. Instrukcija je završena, što znači da su sve akcije u vezi instrukcija obavljene pre nego što se javi trap. Ovo je slučaj, na primer, ako se detektuje premašaj.
2. Instrukcija se prekida, smatra se da operacija nije bila specificirana. Javlja se kada se detektuje premašaj kod zaštite.
3. Instrukcija je poništena - instrukcija se ne izvršava, a PC se inkrementira. Javlja se kao slučaj kada se javi greška u straničenju.
4. Instrukcija se završava - to znači da određeni deo ili sve akcije instrukcije nisu obavljene.

Trenutak generisanja zahteva za prekid kod asinhronih događa se obično javlja u toku izvršenja instrukcije pa su zbog toga i mogućnosti minimalne da se stanje procesora u tom trenutku sačuva. Kod svih procesora kada se primi zahtev za prekid (spoljni) tekuća instrukcija se izvršava pa nakon toga počinje opsluživanje zahteva za prekid (tj. izvršenje prekidnog programa). Ipak, kod nekih instrukcija, kao što su instrukcije za manipulisanje nizovima i vektorima, završetak tih instrukcija će dovesti do suviše velikog odlaganja. Ovo će učiniti da latentnost prekida (vreme od trenutka kada se izda zahtev za prekid, pa sve do izvršenja rutine za opsluživanje prekida) bude neprihvatljivo duga. Zbog toga se ove instrukcije mogu prekidati, a stanje prekinute instrukcije se može obnoviti nakon prekida, tj. ona produžava od tačke gde je bila prekinuta. Alternativno, instrukcije se mogu učiniti da budu restartujućeg tipa, tako da se one mogu ponovo izvršiti nakon završetka prekida. Ovo se odnosi samo na one

instrukcije koje ne menjaju inicijalne uslove koje su esencijalni za tu instrukciju - operacije za pomeranje blokova koje koriste preklapanje ne mogu biti restartujuće. Treća mogućnost je da se dozvoli rad prekidima u određenim (programsko-kontrolisanim) trenucima. Ovo se može koristiti kod vremensko-kritičnih rutina, a da je pri tome njihov kontekst relativno mali.

Identifikacija izvora prekida

Proces koji se priziva kao rezultat generisanog zahteva za prekid izvršava se u ime procesa koji je izdao zahtev za prekid. Kod asinhronih prekida, to može biti bilo koji proces. Opsluživanje zahteva za prekid obično znači da je potrebno identifikovati proces koji je generisao taj zahtev. Razlikujemo dve metode za identifikaciju procesa koji je generisao zahtev za prekid (zove se takođe i izvor prekida) a one su sledeće:

1. **Kružna analiza (polling)** - sistemski program koji se zove program za obradu prekida (interrupt handler) treba da identifikuje izvor prekida. Najjednostavniji način za identifikaciju izvora prekida je kada se svakom mogućem izvoru prekida pridruži marker za generisanje zahteva za prekid. "Interrupt handler" izvršava sekvencu instrukcija, testira (ili puluje) svaki marker redom, sve dok ne ustanovi koji je od markera postavljen. Imajući u vidu da se testiranje obavlja od strane "interrupt handler"-a (koji je program napisan da bude nezavisan od strane uređaja) poželjno je da testiranje stanja markera (preko koga svaki od uređaja izdaje zahtev za prekid) ne bude zavisno od tipa (specifičnosti izvođenja) uređaja. Da bi se to obavilo neophodno je da svaki uređaj poseduje statusni registar kod koga su format reči, pozicija bitova i njihovo značenje za sve uređaje identični. Ako postoji veći broj potencijalnih izvora prekida, proces kružne analize se može ubrzati istovremenim čitanjem svih zahteva za prekid i upisom te informacije u registar za prekid, koristeći pri tome specijalnu U/I komandu. Registar za prekid se može nakon toga analizirati sa ciljem da se ustanovi na kojoj je bit poziciji njegovog sadržaja postavljena jedinica.
2. **Vektorski prekid** - Ovaj način omogućava procesima koji zahtevaju prekid da sami sebe identifikuju u toku operacije priznavanja prekida. Ovakim principom rada eliminiše se vreme koje je potrebno za testiranje i proveru statusa uređaja koji je izdao zahtev za prekid, ali se cena plaća kompleksnijom hardverskom strukturom za detektovanje izvora koji je izdao zahtev za prekid. Po jedan broj, nazvan prekidni vektor, pridružuje se svakom izvoru prekida. Ovaj vektor se koristi da modifikuje ulazne tačke (početnu adesu prekidne rutine) u Interrupt handling rutinu. Selekcija određene ulazne tačke rutine za obradu prekida identifikuje izvor prekida.

4.4.6. Tehnike za obradu izuzetaka kod mikroprocesora MC68020

U daljem tekstu ukazaćemo na tehnike rada koje se koriste za obradu izuzetaka, a važe za mikroporcesor MC68020. U toku objašnjenja koja slede koristićemo terminologiju firme Motorola koja je proizvođač mikroprocesora MC68020.

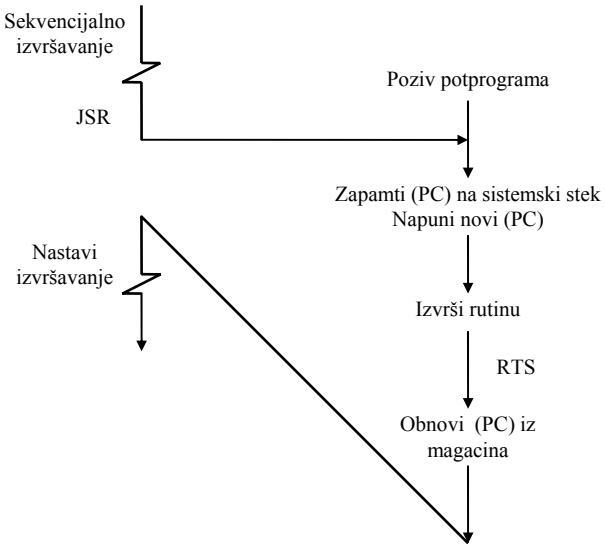
MC68020 koristi magacin za čuvanje informacije u trenutku kada se vrši poziv potprograma ili kada se u toku rada mašine javi izuzetak. U ove izuzetke, kao što je to definisano od strane Motorole, možemo svrstati: trapove, prekide i neke greške koje se prepoznaju od strane CPU-a. Kada se javi bilo koji od ovih događaja normalni tok programa se prekida. Upravljanje se predaje instrukcijama koje su pridružene potprogramu, trapu ili prekidu sve dok se taj zadatak ne izvrši. Upravljanje se nakon završetka standardno vraća narednoj instrukciji u programskoj sekvenci čije je izvršenje bilo istisnuto. Da bi obezbedio korektni nastavak izvršenja programa nakon povratka, CPU smešta informaciju u odgovarajući sistemski magacin. Kod poziva potprograma u magacin se smešta samo PC. Kada se javi izuzetak, sadržaj PC-a, SR-a kao i druge informacije, u saglasnosti sa zahtevima izuzetka, smeštaju se u magacin. Operacije smeštanja i izbavljanja iz magacina se izvode automatski i ne zahtevaju intervenciju programera. Kod MC68020, lokacije u memoriji koje se koriste za korisnički i supervizorski magacin su izdvojene. Kada CPU radi u supervizorskom načinu rada sistemski magacin je supervizorski magacin, a pokazivač magacina je SSP. U korisničkom načinu rada može da se menja samo USP, tj. ne može da se vrši pristup preko SSP-a. U toku izvršenja programa (u jednom trenutku) koristi se samo jedan magacin.

Na slici 4.16a) prikazan je tok upravljanja kada se izvrši poziv potprograma. Kada se prepozna izuzetak (od strane MC68020) prelazak na rutinu za obradu izuzetaka se izvodi automatski od strane CPU-a, kao što je prikazano na slici 4.16b).

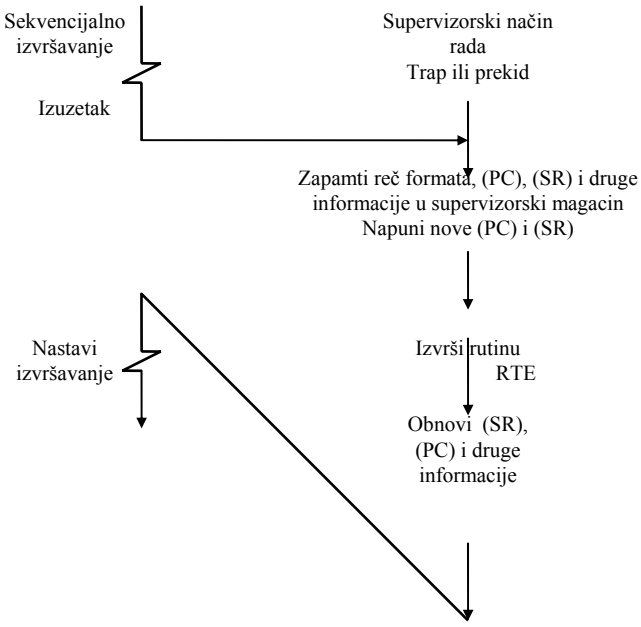
Na slici 4.17 prikazani su sadržaji i načini korišćenja magacina u toku korišćenja poziva potprograma i obrade prekida.

Izuzeci kod MC68020 se mogu grubo podeliti na one koji su:

- uzrokovani instrukcijom (u ovu grupu spadaju i izuzeci generisani usled neobičnih uslova koji se javljaju u toku izvršenja instrukcija) - zovu se trapovi i predstavljaju izuzetne uslove čije je generisanje uslovljeno izvršenjem programa a detektuju se od strane CPU-a.
- uzrokovani spoljnim događajem - sistemski događaji i drugi uslovi koji se iniciraju spoljnim hardverom. Prekidi i izuzeci usled greške na magistrali pripadaju ovoj grupi i prepoznaju se od strane CPU-a.



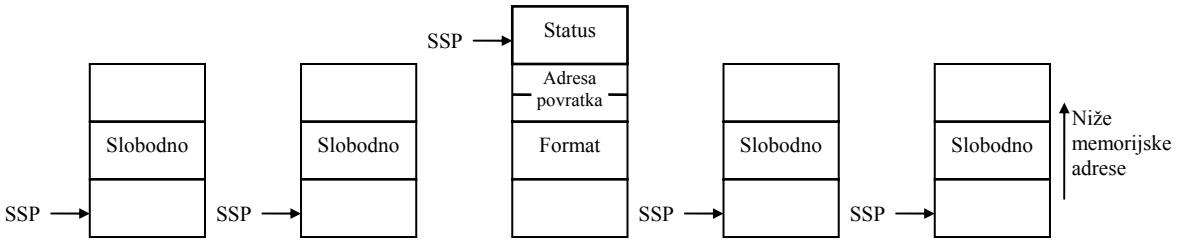
(a) Poziv potprograma.



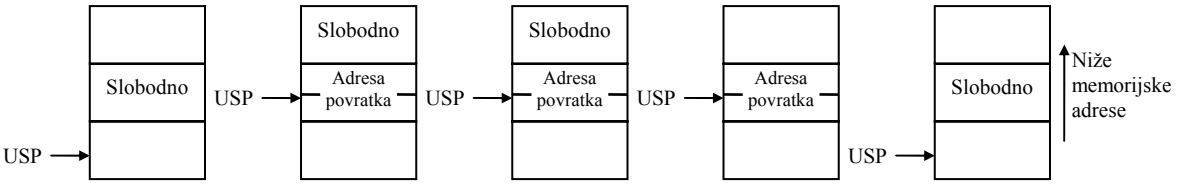
(b) Izuzetak.

Sl. 4.16. Tok upravljanja za vreme izvršenja programa.

U Tabeli 4.10. prikazana je lista izuzetaka i uzrok njihovog generisanja. Tipovi izuzetaka se mogu podeliti na one generisane od strane: TRAP instrukcija, programskih provera, specijalnih izuzetaka, instrukcija koje signaliziraju grešku u toku izvršenja, sistemskih grešaka, uslova koji uključuju izuzetke u toku rada sa koprocesorom, i prekida. Zadnja tri tipa izuzetaka iniciraju usled preuzimanja određenih aktivnosti ili usled akcija spoljnih kola čiji je rad asinhron u odnosu na CPU-ov, a svi ostali se javljaju u toku normalnog programskog izvršenja.



(a) Supervizorski stek



(a) Korisnički stek

Sekvenca izvršenja:

Korisnički program	Potprogram	Prekidna rutina	Potprogram	Korisnički program
(i) Pre poziva	(ii) Za vreme izvršavanja	(iii) Za vreme prekida	(iv) Posle povratka na potprogram	(v) Posle povratka iz potprograma

VREME ⟶

Sl. 4.17. Korišćenje steka za vreme poziva potprograma i obrade prekida.

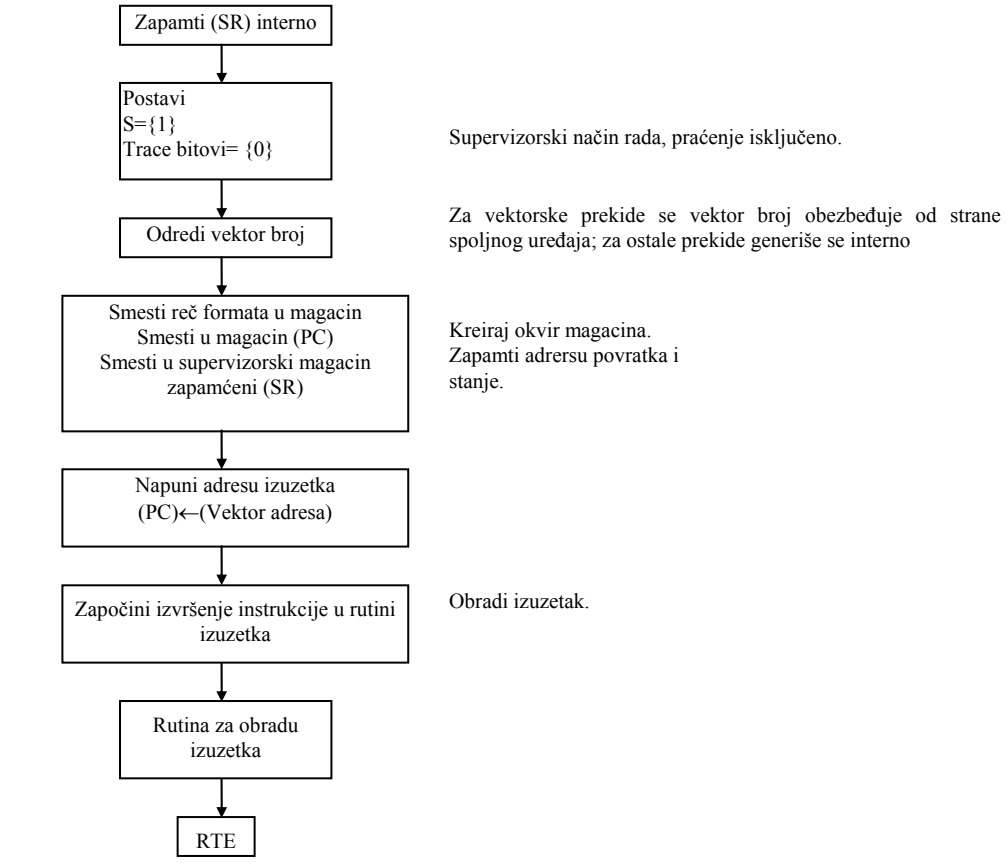
Tab. 4.10. Izuzeci kod MC68020.

Tip	Opis
Trap instrukcija	
TRAP #<N>; N = 0, 1, ..., 15	Šesnaest trap rutina se može definisati i koristiti u programu
Programski testovi	
DIVS, DIVSL, DIVU, DIVUL	Trap ako je delilac nula
CHK, CHK2	Trap ako je vrednost u registru izvan granica
TRAPcc	Trap na jedan od 16 aritmetičkih uslova
TRAPV	Trap na prekoračenje, tj. ako je V={1}
Specijalni izuzeci	
Neimplementirana instrukcija Trap	Tarp ako je kod operacije ili {1010} ili {111} kada nema koprocesra
Izuzeci praćenja	Praćenje posle svake instrukcije ili posle promene toka upravljanja
Prekidne tačke (BKPT)	Trap ako se javi greška magistrale tokom izvršenja BKPT instrukcije
Uslovi greške instrukcije	
Narušavanje privilegije	Trap ako program u korisničkom načinu rada pokuša da izvrši privilegovanu instrukciju
Ilegalne instrukcije	Trap ako reč operacije instrukcije ne predstavlja valjani bit šablon
Adresna greška	Trap ako procesor pokuša da pribavi instrukciju na neparnoj adresi
Sistemske greške i uslovi	Spolja generisani zahtev za izuzetkom
Greška magistrale	Trap ako se pojavi greška tokom izvršenja RTE, CALLM ili RTM ili cpRESTORE*
Greška formata	Uslov dvostruke greške zaustavlja procesor
Halt	
Koprocesorski izuzeci	Izuzetak izazvan aktivnošću koprocesora
Sistem prekida	Automatsko vektorisanje za sedam nivoa prioriteta
Autovektor	192 korisnički definisana prekidna vektora; prioritet određen
Vektorisan	projektom hardvera

Napomena: MC6830 CPU ne prepoznaje CALLM i RTM instrukcije

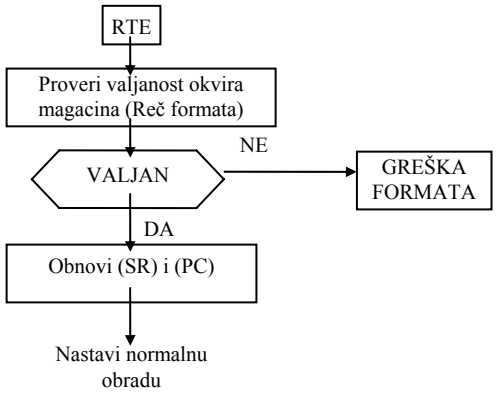
Na slici 4.18 prikazana je procesna sekvenca koja važi za obradu najvećeg broja izuzetaka, isključujući izuzetak "Reset" čiji smo rad već jedanput analizirali.

Vektor brojevi sa pripadajućim memorijskim adresama za svaki od izuzetaka (za MC68020) prikazani su na slici 4.19.



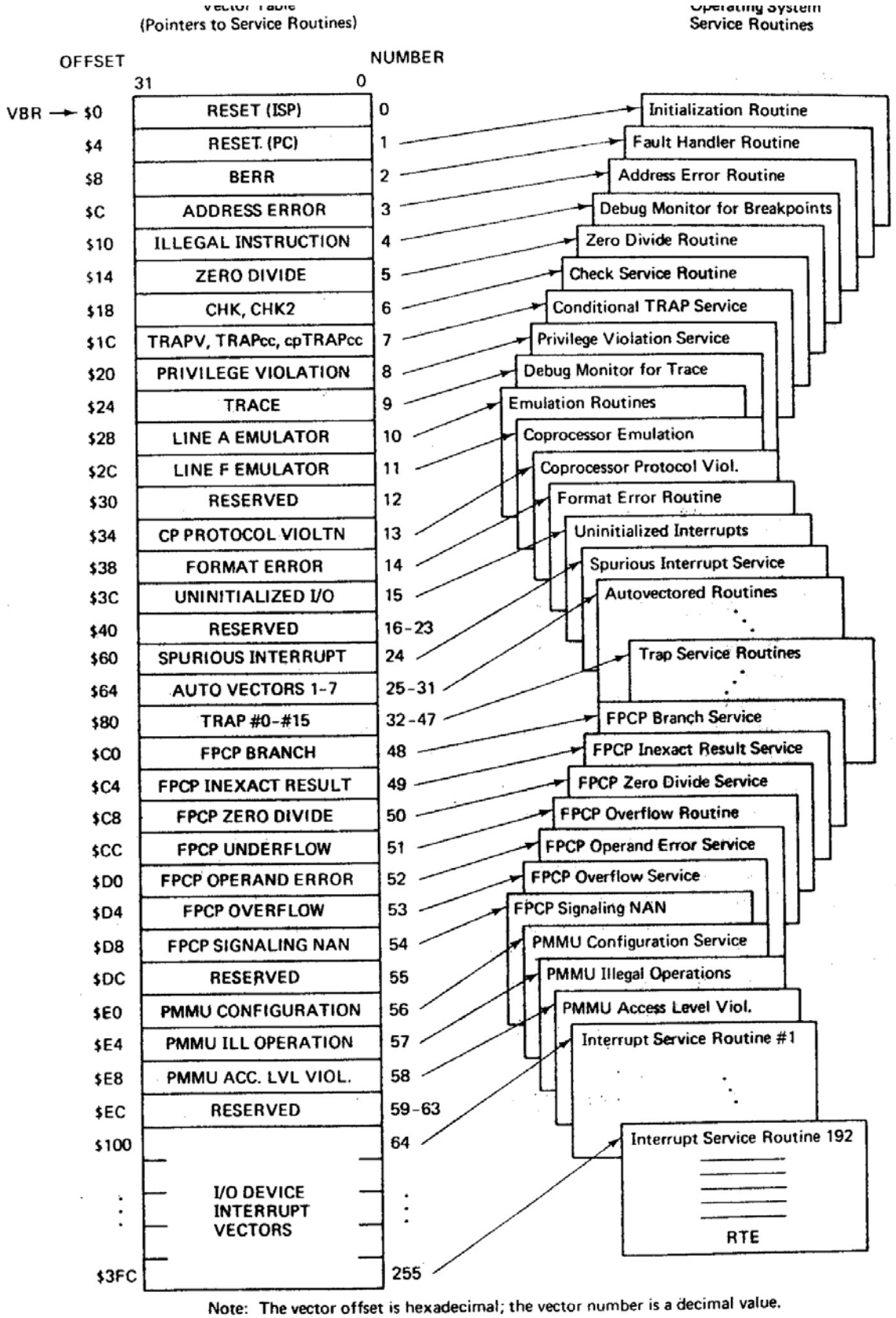
Napomena: Obrada za reset i neke druge izuzetke se delimično razlikuje od ove koja je prikazana.

(a)



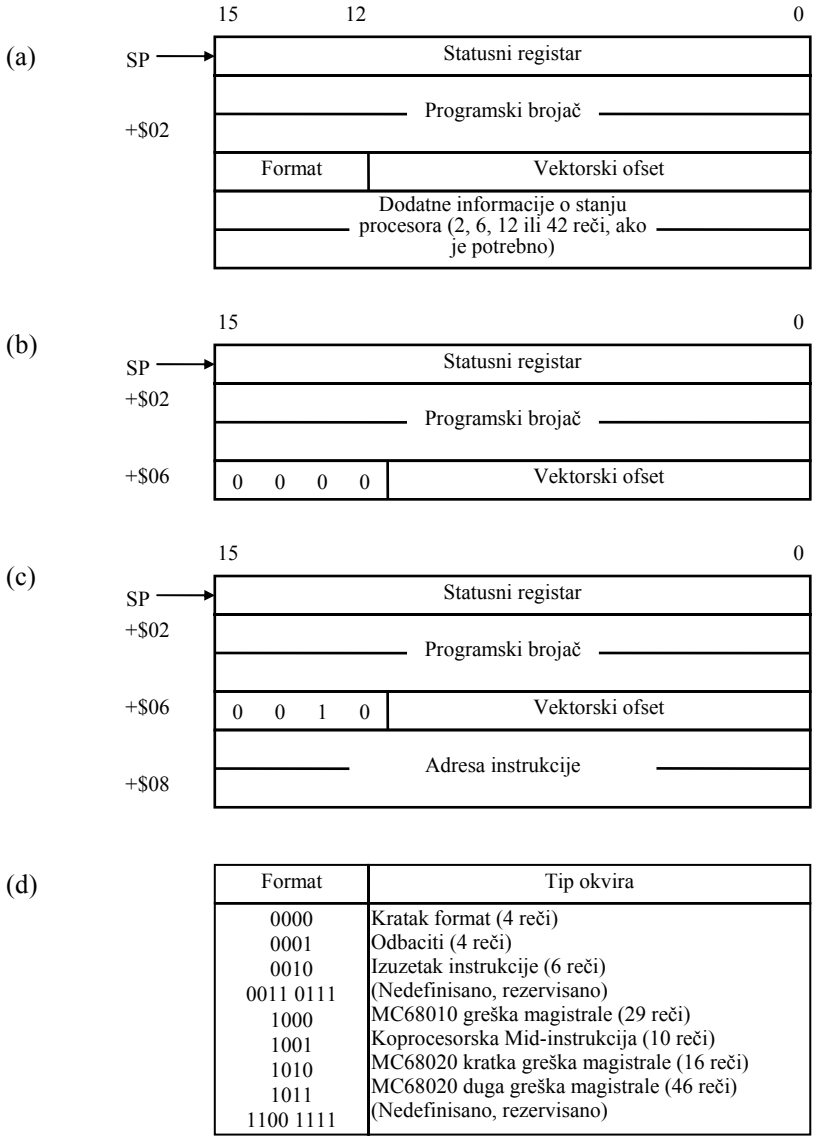
(b)

Sl. 4.18. (a) Sekvenca obrade izuzetka kada se prepozna izuzetak. (b) povratak iz sekvence izuzetka.



Sl. 4.19. Vektorska dodela.

Opšti format okvira magacina koji važi za izuzetke prikazan je na slici 4.20.



Sl. 4.20. (Okvir magacina izuzetka za MC68020; (b) okvir magacina od četiri reči; (c) okvir magacina od šest reči; (d) kodovi formata za okvir magacina izuzetka.

Obrada izuzetaka od strane OS i monitorskog programa

Bilo koji supervizorski program isporučen kao gotov proizvod od strane firme Motorola ili drugih kompanija (odnosi se na sisteme zasnovane na MC68020) sadržaće Rutine za Obradu Izuzetaka (ROI) kao deo sistemskog softvera. Ove rutine se obično zovu standardne ili opšte i prate isporuku svakog sistema. Druga klasa ROI-a se zove ROI specijalne namene. Ove rutine zavise od primene konkretnog računarskog sistema.

Standardni rukovaoci izuzetaka

OS računara zasnovan na MC68020 ima dve kategorije rukovaoca izuzetaka (EH - Exception Handler). U prvu kategoriju spadaju one rutine koje obezbeđuju usluge programu. Pod uslugama se podrazumevaju ulazne i izlazne operacije, detekcija grešaka i druge usluge koje se normalno obrađuju od strane OS-a.

Specifična usluga se zahteva pomoću instrukcija

TRAP #<N>

Konstanta N se koristi da definiše jedan od 16 mogućih zahteva operativnom sistemu. Druga kategorija ROI je deo operativnog sistema i sadrži rutine koje upravljaju radom perifernih uređaja koji se povezuju sa računarom.

EH specifične namene

Kod velikog broja aplikacija, javljaju se specijalni uslovi u toku rada računarskog sistema, pa je shodno tome neophodno realizovati specifične EH. Ove rutine su specifične i zavisne od tipa aplikacije (FFT, matrične operacije i dr.)

Izuzeci uslovljeni instrukcijama TRAP i programskom proverom

Nekoliko izuzetaka, nazvanih trapovi, dostupni su programeru radi upravljanja radom programa. Lista ovih izuzetaka je prikazana u Tabeli 4.11, a uključuje izuzetke koji su uslovljeni od:

a) TRAP instrukcija
b) instrukcija za proveru programskih uslova.

Tab. 4.11. TRAP instrukcije i programske provere.

Tip trapa	Uzrok trapa	Vektorska adresa	Komentari
TRAP instrukcija	Normalno izvršenje	$080 - $0BC	Tipičan metod poziva supervizorskog programa
Programske provere			
DIVS, DIVSL, DIVU, DIVUL	Deljenik jednak nuli	$014	Aritmetičke provere
CHK, CHK2	Vrednost u registru je van granica	$018	Provere granica
TRAPcc	Detektovan logički uslov ″cc″	$01C	Jedan od 16 uslova se može izabrati
TRAPV	Izvršenje sa V={1}	$01C	Trap na aritmetičko prekoračenje

Trap instrukcija

Ova instrukcija se koristi za prenos upravljanja supervizorskom programu u slučaju kada se zahteva supervizorska usluga. TRAP ima sledeći format

TRAP #<vektor>

TRAP prvo uslovljava da se sadržaji PC-a i SR-a smeste u supervizorski magacin. Obrada počinje na adresi specificranoj vektor lokacijom. Vrednost #<vektor> je celobrojna vrednost iz opsega 0-15, a koristi se za izračunavanje heksadecimalne adrese na sledeći način

vektor adresa = 8016 + 4*<vektor>

Izvršenje programa počinje punjenjem PC-a 32-bitnom adresom

(PC) ← (vektor adresa)

Vektor adrese su prikazane u Tabeli 4.12 (lokacije se inicijaliziraju pre izvršenja TRAP-a).

Tab. 4.12. TRAP vektorske adrese.

TRAP instrukcija	#<N>	Vektor adresa (heksadecimalno)
TRAP	#0	80
TRAP	#1	84
TRAP	#2	88
TRAP	#3	8C
TRAP	#4	90
TRAP	#5	94
TRAP	#6	98
TRAP	#7	9C
TRAP	#8	A0
TRAP	#9	A4
TRAP	#10	A8
TRAP	#11	AC
TRAP	#12	B0
TRAP	#13	B4
TRAP	#14	B8
TRAP	#15	BC

Napomena: Vektor broj je decimalan.

Trap deljenje nulom

Određene aritmetičke greške (u aplikacionim programima) se mogu detektovati, a shodno tome i generisati trap od strane CPU-a. Tipičan primer je instrukcija deljenja, kada je delitelj nula, kojom se automatski uzrokuje trap na adresi $14. Najveći broj operativnih sistema završava program kod koga se javlja ovaj tip trapa, jer je dalja aritmetička obrada nakon deljenja nulom nepoželjna. Trap deljenje nulom kreira okvir magacina od 6 reči koji pored ostalog sadrži i adresu instrukcije koja je uslovila trap.

Instrukcije CHK i CHK2

MC68020 poseduje dve instrukcije koje omogućavaju proveru da li je vrednost u registru u okviru zadatog numeričkog opsega. CHK instrukcija proverava vrednost u Dn registru za opseg od 0 do gornje granice. CHK2 testira adresu ili vrednost upisanu u registar Dn sa ciljem da proveri da li se ona nalazi između dve granice. Kod obe instrukcije javlja se trap ako je vrednost u testiranom registru van granica.

Instrukcija CHK

Simbolički oblik ove instrukcije je

CHK.<l> <EA>,<Dn>

Instrukcijom se određuje da li je sadržaj <Dn> između 0 i vrednosti na lokaciji <EA>, i uzrokuje trap ako sadržaj <Dn> nije unutar ovog opsega. Gornja granica koja se čuva u (EA) se tretira kao 16-bitna, celobrojna vrednost u dvoičnom komplementu. Aktivnost operacije se može izraziti kao

IF < (Dn)[l] ≤ (EA), THEN continue
ELSE trap and
setN={1} if (Dn)[l] < 0 or
setN={0} if (Dn)[l] > (EA)

ROI počinje sa adrese na lokaciji $18 ako se javi trap.

Primer 4.4:

Sekvencom instrukcija

```
CHK     #99,D1
MOVE.B  (0,A1,D1.W),D2
```

Javiće se trap ako (D1)[15:0] premašuje 99 decimalno. Ako se u A1 čuva bazna adresa polja dužine 100 elemenata tipa bajt, tada je opseg adresiranja MOVE instrukcije ograničen na vrednost u okviru ovog polja.

Instrukcija CHK2

Format ove instrukcije je sledeći:

CHK2.<l> <EA>,Rn

<EA> definiše memorijsku adresu granice koja se koristi za proveru u poređenju sa vrednošću u registru Rn. Vrednost na koju se ukazuje, nezavisno od toga da li je bajt ili reč, znakovno se proširuje na 32-bitnu vrednost prilikom poređenja. U memoriji, na lokaciji označenoj sa <EA>, donja granica može ukazivati na bajt, reč ili duplu reč u zavisnosti od specifikacije .<l>, a gornja granica mora da sledi u narednoj lokaciji. Poređenje je tipa

(<EA> ≤ (Rn) ≤ (<EA+k>)

gde je k=1,2 ili 4 bajta respektivno.

Efekat izvršenja ove instrukcije sa stanovišta postavljenosti markera uslova identičan je kao i instrukcije CMP2, sa izuzetkom što se u ovom slučaju javlja trap ako se u toku kompariranja javi uslov "van-opsega".

Instrukcije TRAPcc i TRAPV

TRAPcc instrukcija uzrokuje trap kada je specificirani uslov "cc" istinit. Kod TRAPV javlja se trap samo kada je marker uslova V={1}.

TRAPcc instrukcija

Kao što je prikazano u Tabeli 4.13a), TRAPcc instrukcija ima nekoliko asemblersko-jezičnih formi, koje odgovaraju dozvoljenoj sintaksi za asembler firme Motorola MC68020. Uslovi "cc" su definisani u Tabeli 4.13b).

Tab. 4.13. TRAPcc instrukcija.
(a) Sintaksa instrukcije

Sintaksa	Efekat
Tcc ili TRAPcc	TRAP IF ″cc″ = true ELSE sledeća instrukcija
Tpcc.<l> #<data> ili TRAPcc.<l> #<data>	TRAP IF ″cc″ = true ELSE sledeća instrukcija posle ″<data>″

Napomena: <l>=W ili L. Ako je <l>=W, <data> je 16-bitna vrednost. Ako je <l>=L, <data> je 32-bitna vrednost.

(b) Uslovi ″cc″

CC	Prenos obrisan	0100	$\overline{C}$	LS	Manje ili isto	0011	$C+Z$
CS	Prenos postavljen	0101	C	LT	Manje od	1101	$N\cdot\overline{V}+\overline{N}\cdot V$
EQ	Jednako	0111	Z	MI	Minus	1011	N
F	Nikad tačno	0001	0	NE	Nejednako	0110	$\overline{Z}$
GE	Veće ili jednako	1100	$N\cdot V+\overline{N}\cdot\overline{V}$	PL	Plus	1010	$\overline{N}$
GT	Veće od	1110	$N\cdot V\cdot Z+\overline{N}\cdot\overline{V}\cdot\overline{Z}$	T	Uvek tačno	0000	1
HI	Više	0010	$\overline{C}\cdot\overline{Z}$	VC	Prekoračenje obrisano	1000	$\overline{V}$
LE	Manje ili jednako	1111	$Z+N\cdot\overline{V}+\overline{N}\cdot V$	VS	Prekoračenje postavljeno	1001	V

Napomena: Za neke asemblere, ″CS″ = ″LO″ a ″CC″=″HS″.

Primer 4.5:

Instrukcije

```
TEQ      ; Trap ako je Z={1}
```

i

```
TRAPEQ
```

su identične za asembler firme Motorola.

Instrukcija "Trap if minus"

```
TPMI.W          #$DE
```

kada se izvršava uzrokuje trap ako je N={1}. Ako se javi trap u okviru magacina se čuva šest reči (isto kao i kod CHK instrukcije). Na lokaciji (SP)+8 memoriše se adresa trap instrukcije.

Instrukcija TRAPV

Instrukcija TRAPV postoji u skupu instrukcija mikroprocesora MC68020 sa ciljem da obezbedi kompatibilnost sa programima napisanim za 16-bitni mikroprocesor MC68000.

Ekvivalentna instrukcija mikroprocesora MC68020 je TRAPVS (Trap if V is Set), i koristi se kod onih programa mikroprocesora MC68020 gde se kompatibilnost ne zahteva.

Neimplementirane instrukcije TRAP, TRACE i BREAKPOINTS

Kod MC68020 postoji nekoliko specijalnih izuzetaka koji se koriste za otklanjanje grešaka u programu (dibagiranje). Ovi izuzeci se tipično koriste za razvoj i testiranje bilo kog specijalizovanog hardvera i softvera za datu aplikaciju. Neimplementirana trap instrukcija je posebno pogodna za hardversku i softversku emulaciju. Emulacija označava (u ovom slučaju) da je izvršenje programa ciljnog računara uključujući i obradu spoljnih događaja (kao što su ulaz-izlaz, prekid i dr.) što je moguće realističnije realizovano pomoću druge mašine. Sa druge strane, simulator je program koji na drugoj mašini izvršava samo program ciljnog računara. BKPT (Breakpoint) i izuzetak trasiranja koriste se da olakšaju posao programeru koji je u vezi sa testiranjem i dibagiranjem programa. Prekidne tačke u programu moguće je postaviti koristeći instrukciju ILEGAL.

Neimplementirana instrukcija Trap

CPU mikroprocesora MC68020 prepoznaje instrukcije čija prva četiri bita [15:12] imaju vrednost {1010}($A) ili {1111}($F), kao neimplementirane instrukcije. Ove instrukcije su poznate kao A-linijska i F-linijska instrukcija, respektivno.

A-linijska instrukcija uzrokuje izvršenje ROI definisane vektorom 10 na adresi $28, a u najvećem broju slučajeva koristi se za emulaciju specijalnih instrukcija pa čak i hardverskih mogućnosti drugih računara. Ostalih 12

bitova ove instrukcije koriste se za izbor različitih opcija u toku izvršenja ROI-a. Kod najvećeg broja aplikacija A-linijska instrukcija se javlja kao dodatna "makroinstrukcija" koja se pridružuje skupu naredbi MC68020.

U asemblerskom programu, iskaz

```
DC.W    $AXXX
```

gde su XXX proizvoljne tri heksadecimalne cifre uzrokovaće trap. Kada se javi trap formira se okvir magacina od četiri reči u kome se čuva sadržaj statusnog registra, adresa neimplementirane instrukcije i format reči.

Adresa dobijena instrukcijom oblika

```
MOVEA.L    (2,SP),A1
```

preskače preko sadržaja SR-a koji je smešten u magacin i puni A1 na vrednost koja odgovara memorisanoj vrednosti PC-a. Ova vrednost se može koristiti za lociranje neimplementirane instrukcije, njeno dekodiranje i izbor opcije za ROI. Koristeći memorijsko indirektno adresiranje, iskazom

```
MOVE.W    ([2,SP]),D1
```

moguće je preneti A-linijsku instrukciju u D1 bez korišćenja An. Korišćenje F-linijske instrukcije slično je onoj za A-linijsku. F-linijska instrukcija se tipično koristi za emulaciju koprocesora kada on nije prisutan. Ako procesor prepozna da F-linijska instrukcija nije važeća koprocesorska instrukcija, generisaće se neimplementirana instrukcija izuzetka. ROI u tom slučaju može u zavisnosti od rešenja da obavi bilo kakvu funkciju.

Ako je koprocesor prisutan u sistemu (tj. instaliran je), CPU predaje F-linijsku instrukciju koprocesoru (Ako se važeća koprocesorska instrukcija izvršava na sistemu zasnovanom na MC68020 kod koga nije instaliran koprocesor, CPU prima "bus-error-signal" od spoljnih kola kada CPU pokuša da pristupi nepostojećem koprocesoru. Ovo uzrokuje pojavu neimplementirane instrukcije izuzetka).

Izuzeci kod trasiranja

U toku testiranja programa koga razvijamo, često je pogodno da programer uslovljava CPU da izvrši jednu instrukciju ili kratku sekvencu instrukcija, iza koje sledi prikazivanje insformacije na displeju, koja će se koristiti u cilju dibagiranja. Kod MC68020 postoje dva tipa izuzetaka koji su važeći za trasiranje, a poznati su kao:

- "single-instruction" trasiranje (instrukcija po instrukciju),
- trasiranje koje se javlja kod promene toka programskog izvršenja (change of flow).

Trasiranje se može izvesti u korisničkom i supervizorskom načinu rada. Bitovi trasiranja (T0 i T1) koji pripadaju SR-u mogu se menjati samo u supervizorskom načinu rada. OS ili monitor program može dozvoliti rad "single-instruction" trasiranju izvršenjem instrukcije

```
ORI    [000,SR ; postavlja T1={1}
```

Dozvola rada trasiranja tipa "change of flow" vrši se instrukcijom

```
ORI    `000,SR ; postavlja T0={1}
```

Kada se izuzetak trasiranja prepozna od strane CPU-a, a to se dešava nakon izvršenja instrukcije koja je uslovila trasiranje, izvršava se ROI tipa trasiranje koja koristi vektor 9 na adresi $024. Još na početku ROI-a CPU brani dalji rad trasiranja (trasiranjem se kreira okvir magacina obima četiri reči) pre nego što toj rutini preda upravljanje. PC vrednost koja je smeštena u magacinu na adresi (SP)+2 predstavlja adresu naredne instrukcije prekinutog programa. Ova instrukcija se izvršava posle instrukcije RTE koja je zadnja instrukcija ROI-a trasiranje.

Ako se u toku trasiranja javi prekid, prvo se obrađuje izuzetak trasiranja a zatim prekid, tj. kreira se prvo okvir magacina za trasiranje a zatim za prekid. Ali kad se prelazi na izvršenje prvo se izvršava prekidna rutina. Ako je trasiranje dozvoljeno kada se izvršava nelegalno implementirana instrukcija, trasiranje se može realizovati, jer takva instrukcija ne postoji.

Kada je T1={1}, trasiranje tipa "change of flow" se obavlja pod sledećim uslovima:

a) instrukcijom se uslovljava nesekvencijalno ažuriranje PC-a,
b) instrukcija modifikuje SR
c) instrukcija koprocesora uzrokuje nesekvencijalno inkrementiranje PC-a.

Obično se rutine za trasiranje projektovane za prikaz sdadržaja CPU-ovih registara i memorijskih lokacvija koje prate izvršenje programa koji se trasira.

Prekidne tačke i instrukcija BKPT

Opcija trasiranja omogućava da se normalno izvršenje programa suspenduje, kada se javi određeni uslov. Drugim rečima, izuzetak trasiranja se javlja nakon izvršenja samo jedne instrukcije ako je SR[15]={1}, ili nakon instrukcije koja je uzrokovala promenu toka programa ako je SR[14]={1}. U najvećem broju slučajeva programer ili projektant hardvera žele da zaustave normalno izvršenje programa kada CPU adresira određenu lokaciju u memoriji. Ako lokacija sadrži instrukciju, za program se kaže da se izvršio do prekidne tačke. U prekidnoj tački, ROI obično prikazuje, na CRT displeju, sadržaj registara i drugim informacija koje su od interesa za program koji se testira.

Kada u sistemu (zasnovanom na MC68020) nije instalirana MC68851 MMU postoji nekoliko načina za definisanje prekidnih tačaka u programu. Prvi način se zasniva na korišćenju instrukcije

ILLEGAL

čija je opreč $4AFC. Ova instrukcija uzrokuje izuzetak tipa ilegalne instrukcije i koristi vektor 4 sa lokacije $010. Drugi način se zasniva na korišćenju instrukcije TRAP #<N>. Bilo koja od ovih instrukcija (ILLEGAL ili TRAP) usloviće izuzetak ako se umetne (insertuje) u niz instrukcija programa koji se debagira.

Mikroprocesor MC68020 poseduje i BKPT (Breakpoint) instrukciju koja zahteva da spoljni hardver postavi i odstarni prekidnu tačku iz programa koji se testira. Ova instrukcija se najčešće koristi ako je u računarski sistem ugrađena MC68851 MMU, jer MC68851 ima mogućnost da obavlja operacije koje su neophodne za podršku rada instrukcije BKPT.

Programske greške koje uzrokuju trapove

MC68020 je projektova da štiti računarski sistem od grešaka koje mogu usloviti nepredvidljivo ponašanje. U Tabeli 4.14 izlistane su programske greške koje uzrokuju trap. Trapovi tipa privilegovani prekršaj, ilegalna instrukcija i adresna greška, javljaju se u toku debagiranja programa. Operativni sistem obično prekida izvršenje programa kod koga se javlja greška.

Privilegovani prekršaj

Ako program koji se izvršava u korisničkom načinu rada pokuša da izvrši jednu od privilegovanih instrukcija datih u Tabeli 4.14, javiće se izuzetak koji koristi vektor 8 na lokaciji $020.. Formira se okvir magacina obima četiri reči a PC ukazuje na lokaciju instrukcije koja je uzrokovala prekršaj.

Tab. 4.14. Greške koje izazivaju trap.

Greška	Uzrok	Komentari
Narušavanje privilegije	U korisničkom načinu rada pokušaj da se izvrši privilegovana instrukcija	Ako je S={0}, pokušaj da se izvrši: ANDI, EORI, MOVE ili ORI sa SR; MOVE iz SR; MOVE u USP; MOVEC, MOVES; RESET; TE; STOP; cpRESTORE; cpSAVE
Ilegalna instrukcija	Bit šablon koda operacije nije prepoznat	PC vrednost u steku je adresa ilegalne instrukcije
Adresna greška	Pokušaj da se pribavi instrukcija sa neparne adrese	Okvir magacina izuzetka Greška magistrale se kreira

Ilegalna instrukcija

Trap iliegana instrukcija sa vektorom na lokaciji $10 koristi se da zaštiti sistem od efekata nelegalnog pribavljanja mašinskog koda ili lokalizovane memorijske greške.

Adresna greška

Ako CPU pristupa instrukciji na neparnoj adresi, javlja se izuzetak tipa adresna greška. Informacija koja se smešta u magacin može se koristiti u dijagnostičke svrhe.

Sistemske greške i uslovi

CPU prepoznaje veći broj sistemskih grešaka i uslova koji se koriste za upravljanje i zaštitu RS-a. Jedna od najvažnijih je izuzetak greška-na-magistrali koja je uzrokovana spoljnim kolima. Drugi tipovi su izuzetak greška-u-formatu, stanje zastoj, i izuzetak generisan od strane koprocesora.

Izuzetak greška na magistrali

Ovaj tip izuzetka signalizira se CPU-u preko BUS-ERROR signalne linije i aktivira se od strane spoljnog hardvera ili MC68852 MMU. Uzrok signala greška-na-magistrali (BUS-ERROR) nije standardizovan sa izuzetkom onog uzroka koji se odnosi na čip MC68851. U opštem slučaju ova signalna linija se aktivira od strane spoljnog kola kada se javi greška ili specijalni uslov. Signal BUS-ERROR ukazuje da spoljno kolo ne može da kompletira instrukciju CPU-a koja se tekuće izvršava. Kada CPU izvršava instrukciju, ona očekuje odziv od strane spoljnjeg kola koje ukazuje da je uređaj priznao instrukciju i može da kompletira zahtevanu obradu. Ako se uređaj ne odazove, ili ako se detektuje greška u instrukciji, uređaj aktivira signal BUS-ERROR. U Tabeli 4.15 prikazane su tipične situacije koje se mogu javiti i aktivirati signalnu liniju BUS-ERROR.

Tab. 4.15. Primeri upotrebe signala greška-na-magistrali.

Situacijia	Uzrok greške magistrale	Akcija CPU-a
Nema odziva memorijske lokacije ili periferalnog uređaja	Watchdog tajmer	Izuzetak greška-na-magistralil
Memorijska greška	Kola za ispravljanje greške	Izuzetak greška-na-magistralil
MC68851 detektovane greške ili uslovi	MC6851 jedinica za upravljanje memorijom	Izuzetak greška-na-magistralil
Narušavanje privilegije		
Greška stranice u sistemu virtuelne memorije		
Pokušaj upisa na zaštićenu stranicu		
Narušavanje nivoa pristupa (CALLM)		
Nema koprocesora (emulacija koprocesora)	Watchdog tajmer	Izuzetak F-linijskog emulatora
Greška tokom izvršenja koprocesorske instrukcije	Kola specijalne namene	Izuzetak greška-na-magistralil
Ciklus prihvatanja prekidne tačke	MC6851	Izuzetak ilegalne instrukcije kada se preduzme prekidna tačka
Ciklus prihvatanja prekida	Kola specijalne namene	Izuztak lažnog prekida

Greška formata

U toku izvršenja instrukcije RTE, polje FORMAT CODE, memorisano u magacinu na lokaciji (SP)+6, se proverava zbog validnosti. Ako FORMAT CODE nije korektan generiše se izuzetak greška-formata. Greška formata označava da su sadržaji magacina ili pokazivača magacina promenjeni na nekorektan način od trenutka kada je izuzetak prepoznat do trenutka izvršenja instrukcije RTE.

Stanje zastoj

U ovo stanje CPU ulazi kada ne može više da produži sa izvršenjem. U zastoj se ulazi kada se u sistemu detektuje katastrofalna greška. Kada uđe u ovo stanje CPU se može restartovati jedino preko linije RESET od strane spoljnih kola. Ako se javi greška na magistrali u toku obrade izuzetka tipa adresna greška, greška na magistrali, ili reset, CPU ulazi u stanje zastoj (halted).

Izuzeci kod rada sa koprocesorom

Greške se mogu javiti i kod rada sa koprocesorom. tipični izuzeci su oni koji koriste vektor 13, vektor 7 (uslovljen koprocesorskom cpTRAPcc instrukcijom), i izuzeci koji koriste vektore od 48 do 54. O ovim tipovima grešaka biće dato viće detalja kasnije u delu koji se odnosi na princip rada koprocesora.

Obrada prekida kod MC68020

Prekidni sistem mikroprocesora MC68020 omogućava spoljnjem uređaju da prekine tekuće izvršenje programa i preda upravljanje rutini za obradu prekida (IHR - Interrupt Handling Routine). Zahtev za prekid od strane spoljnjog uređaja može se javiti na jednom od sedam nivoa prioriteta, a određuje se na osnovu vrednosti koja se istovremeno dovodi na sve tri signalne linije CPU-a, a to su IR0-IR2. Prekidi su uređeni po prioritetu pri čemu je nivo 1 najniži, a nivo 7 najviši. Ovi prioriteti omogućavaju da rutina koja se obrađuje na nižem nivou prioriteta bude prekinuta od strane zahteva za prekid višeg nivoa prioriteta, Nakon što se najviši nivo obrade prekida završi, upravljanje se predaje prekidnoj rutini nižeg nivoa koja je čekala na izvršenje. Kada se svi zahtevi za prekid obrade upravljanje se vraća programu koji je bio prekinut.

Sa procesorske tačke gledišta, prekid je spoljašnje generisani zahtev za obradu izuzetka. Zahtev za prekid se može posmatrati kao : a) aktivan, b) aktivan je ali nije uslužen (pending), i c) zabranjen.

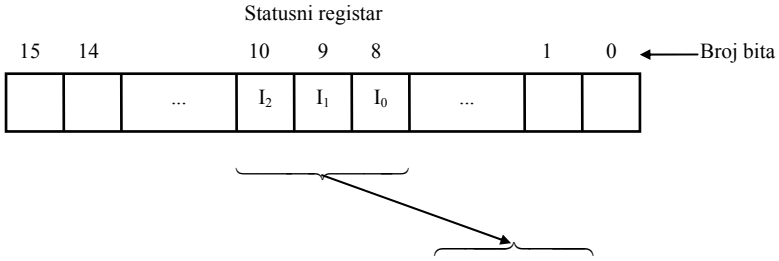
Aktivni zahtev se obrađuje odmah nakon završetka bilo koje instrukcije koja se trenutno izvršava pod uslovom da ne postoji zahtev za obradu izuzetka višeg nivoa. Zahtev je "pending" ako procesor trenutno obrađuje izuzetak višeg nivoa. "Pending" zahtev biće uslužen kada se završi obrada višeg nivoa pod uslovom da procesor uđe u stanje zastoja. Ako je prekidni nivo zabranjen, zahtev za prekid na tom nivou se ignoriše, sve dok se nivou ne dozvoli rad, što se izvodi promenom prekidne maske u (SR)[10:8].

Sistem prekida se inicijalizira od strane supervizorskog programa u toku faze inicijalizacije punjenjem početnih adresa za svaku prekidnu rutinu u odgovarajuću lokaciju u vektor tabeli. Inicijalizacija se obavlja tako što

su svi nivoi prioriteta zabranjeni sa izuzetkom nivoa 7 koji se ne može zabraniti. Prekidima se dozvoljava rad pre nego što se vrši predaja upravljanja sa supervizorskog načina rada na aplikacione programe.

Izgled bitova maske prekida prikazan je na sl. 4.21.

Vektorska tabela svih prekida prikazana je u Tab. 4.16.



Zabranjeni (maskirani) nivoi	Maska prekida		
	I_2	I_1	I_0
1-6	1	1	1
1-6	1	1	0
1-5	1	0	1
1-4	1	0	0
1-3	0	1	1
1-2	0	1	0
1	0	0	1
Svi nivoi dozvoljeni	0	0	0

Sl. 4.21. Maska prekida za MC68020.

Tab. 4.16. Vektorska tabela za prekidne rutine.

Vektor broj (decimalno)	Memorijska lokacija (hksadecimalno)	Naziv
15	$003C	Vektor neinicijalizovanog prekida
24	$0060	Vektor lažnog prekida
25	$0064	Autovektor nivoa 1
26	$0068	Autovektor nivoa 2
27	$006C	Autovektor nivoa 3
28	$0070	Autovektor nivoa 4
29	$0074	Autovektor nivoa 5
30	$0078	Autovektor nivoa 6
31	$007C	Autovektor nivoa 7
. . .	. . .	. . .
64	$0100	Vektor korisničkog prekida 1
65	$0104	Vektor korisničkog prekida 2
. . .	. . .	. . .
255	$03FC	Vektor korisničkog prekida 192

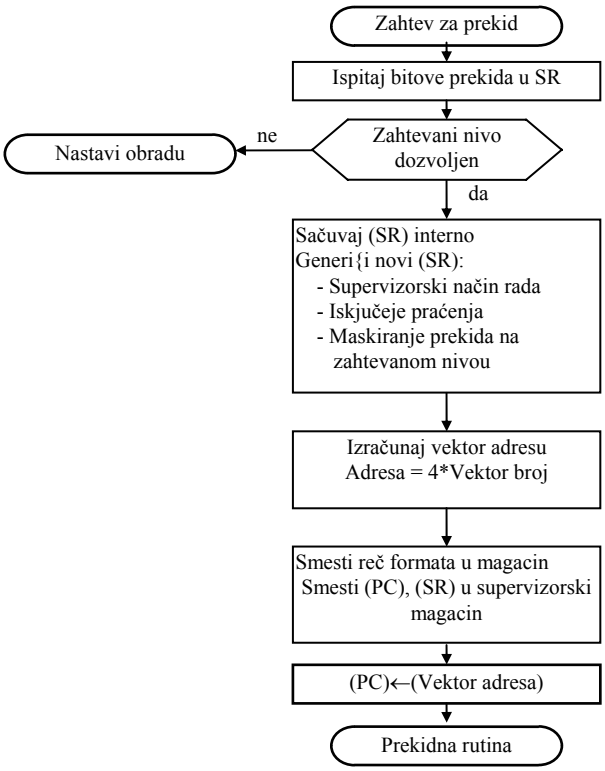
Napomene:
1. Vektor 15 treba da se obezbedi od strane neinicijalizovanog spoljnog uređaja ako CPU zahteva vektor broj.
2. Lažni prekid se javlja kada CPU otkrije grešku tokom obrade prekida.

Lažni (*spurious*) prekidni vektor koristi se kada procesor prepozna zahtev za prekid ali postoji spoljni uslov greške (greška na magistrali od strane spoljnjeg uređaja koja se javlja u toku zahteva za prekid uzrokovaće

lažni prekid). Kao što se vidi iz Tabele 4.16, prekidi su klasifikovani u dve grupe i to a) autovektorski i b) korisnički. Izbor između ova dva načina rada za sistem prekida određen je u potpunosti od strane spoljnih kola. U oba slučaja adresa prekida izračunava se kada se vektor broj pomnoži sa četiri. Kod MC68020 postoje ugrađena dva supervizorska pokazivača magacina, označeni kao ISP (Interrupt Stack Pointer) i MSP (Mater Stack Pointer). Kada je M={0} (SR[12]={0}) ISP je pokazivač aktivnog supervizorskog magacina za sve tipove izuzetaka. Kada OS postavi M=š1ć aktivira se master magacin, a kao pokazivač se koristi MSP. Magacin za prekide se takođe koristi za obradu prekida. Nezavisno od toga što su dostupna dva magacina, samo se sa jednim magacinom može manipulisati u jednom trenutku od strane programa.

Obrada prekida koristeći magacin prekida

Kada procesor izvršava instrukcije u normalnom stanju, a zahtev za prekid se prizna, obavlja se sekvenca događaja u kojoj se vrši prenos upravljanja na određenu prekidnu rutinu. Ovom rutinom vrši se obrada prekida, a na kraju ponovo vraća upravljanje prekinutom programu. Sekvenca događaja je prikazana na slici 4.22.



Sl. 4.22.

ISP i MSP

Kada je M={1}, izuzetak uslovljava da se kreira odgovarajući master okvir u master magacinu. Postavljanje M bita ne utiče na CPU-ov privilegovani način rada, ali korišćenje master magacina omogućava OS-u da izdvoji programsko zavisne izuzetke od prekida.

Okviri magacina i prioriteti izuzetaka

Analizirajmo sada neke detalje koji se odnose na obradu izuzetaka, a nakon toga prioritete izuzetaka.

Okviri magacina

Okvir magacina obima od četiri do 46 reči se kreira kada se prepozna izuzetak.

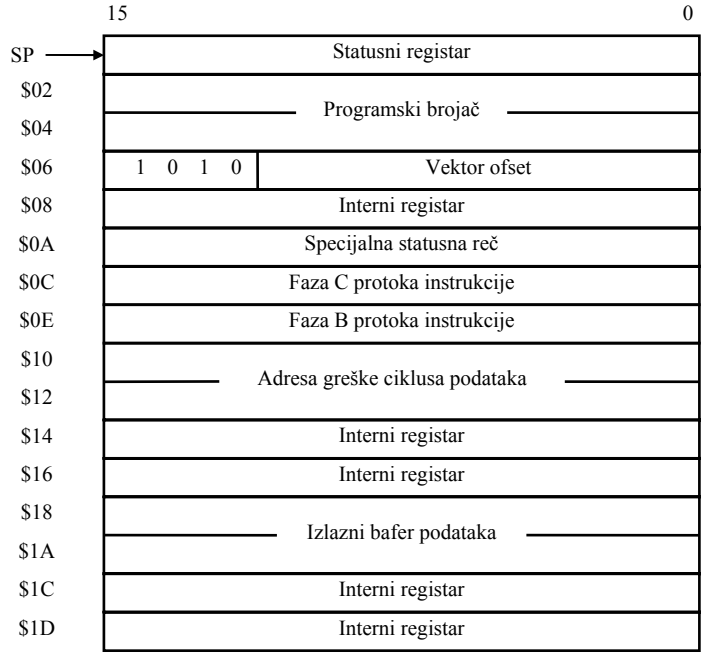
Okvir magacina obima četiri reči kreira se kod prekida, greške formata, TRAP instrukcija, nelegalne instrukcije, A i F linijski emulator trap, privilegovanih prkršaja i određenih koprocesorskih izuzetaka.

Okvir magacina obima 6 reči sadrži adresu instrukcije koja je uzrokovala grešku, kao i sadržaje PC-a i SR-a. Ovaj okvir se kreira od strane CHK, CHK2, cpTRAPcc, TRAPcc, TRAPV, trasiranja, deljenja nulom i određenih koprocesorskih izuzetaka. Ako CPU na početku izvršenja instrukcije detektuje grešku na magistrali kreira se okvir magacina obima 16 reči (slika 4.23).

Ako se javi greška na magistrali u toku izvršenja instrukcije kreira se okvir magacina obima 46 reči (slika 4.24). U ovom slučaju CPU smešta u magacin celokupno svoje stanje, kako bi bio sposoban da produži sa izvršenjem prekinute instrukcije nakon završetka rutine za obradu greške. Moguće je i formiranje okvira magacina obima 10 reči kod određenih tipova koprocesorskih izuzetaka.

Prioritet izuzetaka

Izuzeci se mogu kategorisati u zavisnosti od prioriteta koji su fiksno izvedeni u strukturi CPU-a i ne mogu se menjati. Prioritet definisan od strane Motorola prikazan je u Tab. 4.17.

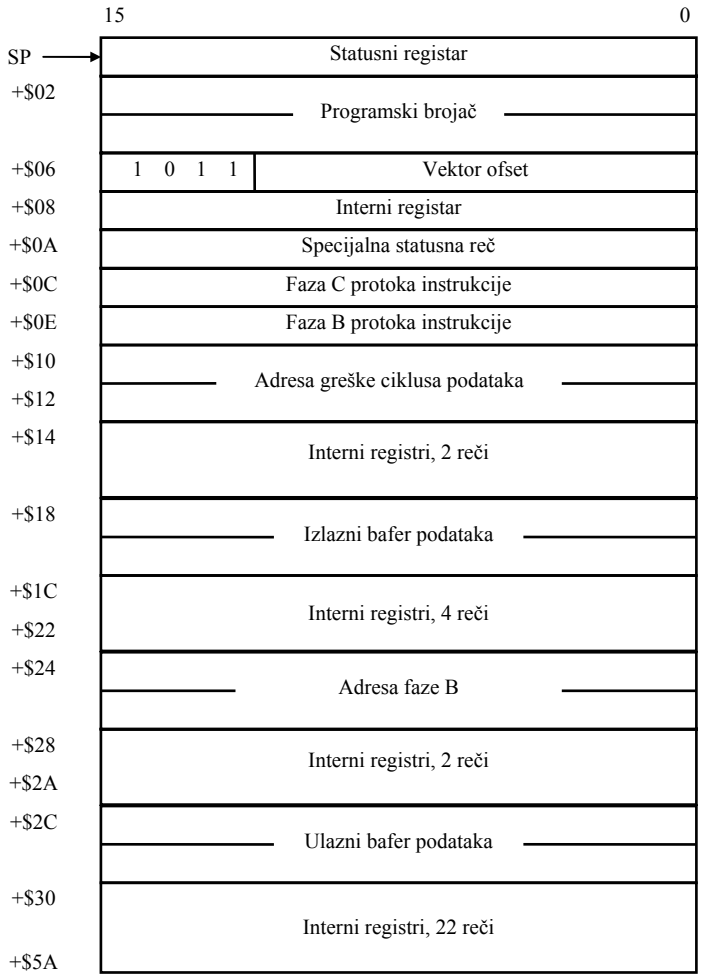


Sl. 4.23. Kratki okvir magacina za grešku ciklusa magistrale.

Tab. 4.17. Prioritet izuzetaka.

Prioritet grupe	Izuzetak i relativni prioritet	Karakteristike
0	0.0-Reset	Prekida svaku vrstu obrade (instrukciju ili izuzetak) i ne pamti stari kontekst
1	1.0-Adresna greška	Suspenduje obradu (instrukciju ili izuzetak) i pamti interni kontekst
	1.1-Greška magistrale	
2	2.0-BKPT #<N>, CALLM, CHK, CHK2, cp mid-instrukcija, narušavanje cp protokola, cpTRAPcc, deljenje nulom, RTE, RTM, TRAP #<N>, TRAPV	Obrada izuzetka je deo izvršenja instrukcije
3	3.0-Ilegalna instrukcija, neimplementirana instrukcija, narušavanje privilegije, cp preinstrukcija	Obrada izuzetka počinje pre nego što se izvrši instrukcija
4	4.0-cp postinstrukcija	Obrada izuzetka počinje kada se tekuća instrukcija ili prethodna obrada izuzetka kompletira
	4.1-Praćenje	
	4.2-Prekid	

Napomena: 0.0 je najviši prioritet, 4.2 je najniži prioritet.



Sl.4.24. Dugi okvir magacina za grešku ciklusa magistrale.