R A Z D E L A V A M O D U L A S T A T E X

V Ljubljani, 1.5.1991

AVTOR:
Vrhovec Borut, dipl.ing.

K A Z A L O

1. A V T O M A T S T A N J

Avtomat stanj omogoča prehode med pozameznimi režimi in podrežimi (stanji) sistema LJUMO ter obdeluje dogodke (reagira na pritisnjene tipke).

Avtomat stanj je organiziran skladovno. Njegova osnova je področje **state_stack,** ki je rezervirano za shranjevanje podatkov o stanjih skozi katera smo šli v nekem glavnem načinu dela (npr. avtomatskem izvajanju). Ko pridemo v novo stanje sistema, se na state_stack shranijo naslednji podatki (nanje kaže kazalec tekočega stanja **curr_state**):

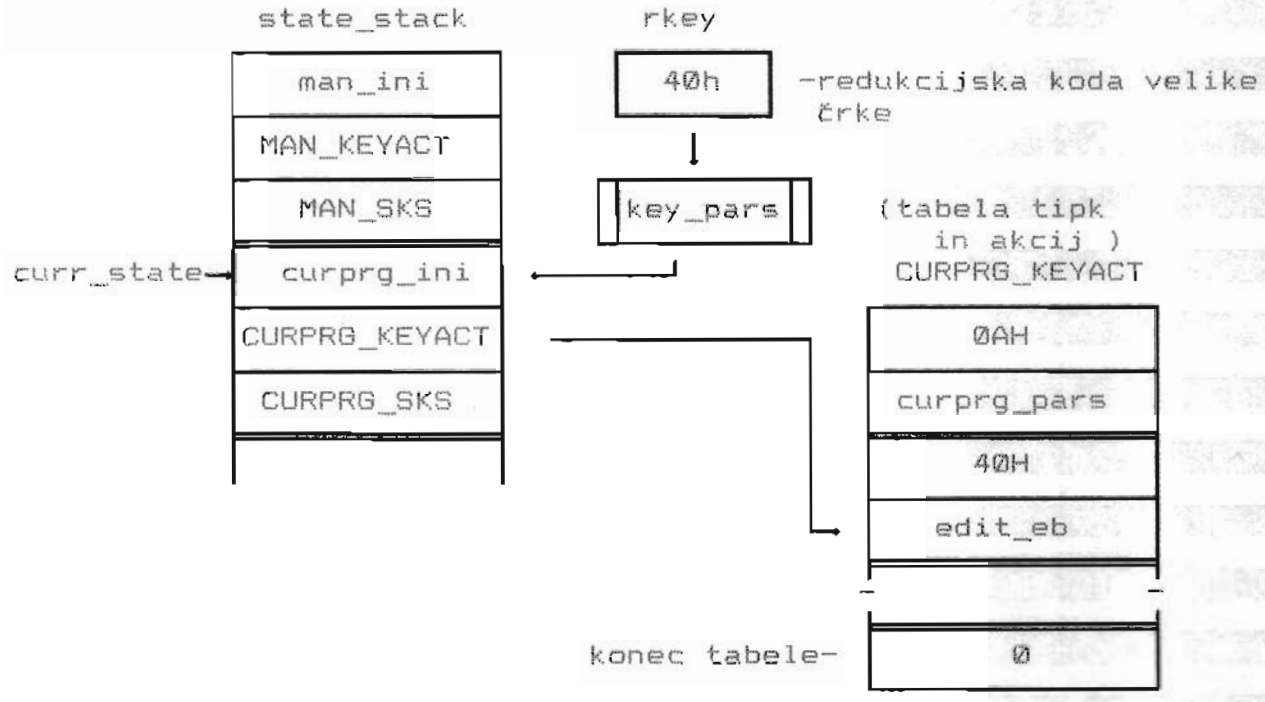
- kazalec na inicializacijsko rutino tega stanja , ki jo potrebujemo pri vračanju iz podstanja,
- kazalec na tabelo dovoljenih tipk in ustreznih akcijskih rutin, ki jo potrebujemo za obdelavo pritisnjenih tipk,
- kazalec na tabelo softkey stringov, ki so lahko imena podstanj ali pa opcije dialoga, in pripadajočih rutin.

Tabela dovoljenih tipk in akcijskih rutin stanja je sestavljena iz redukcijskih kod dovoljenih tipk in kazalcev na akcijske rutine teh tipk. Konec tabele označuje vrednost 0.

Tabela softkey tipk in rutin stanja je sestavljena iz ASCII napisov softkey tipk in kazalcev na ustrezne rutine. Tabela mora biti sestavljena iz blokov po pet sks struktur, ker ima sistem pet softkey tipk. Konec tabele označuje vrednost 0, na mestu sk stringa. Kazalec na rutino, ki pripada prazni tipki (prazen napis) je enak 0. Ob pritisku prazne tipke ni akcije.

OBDELAVA PRITISNJENIH TIPK

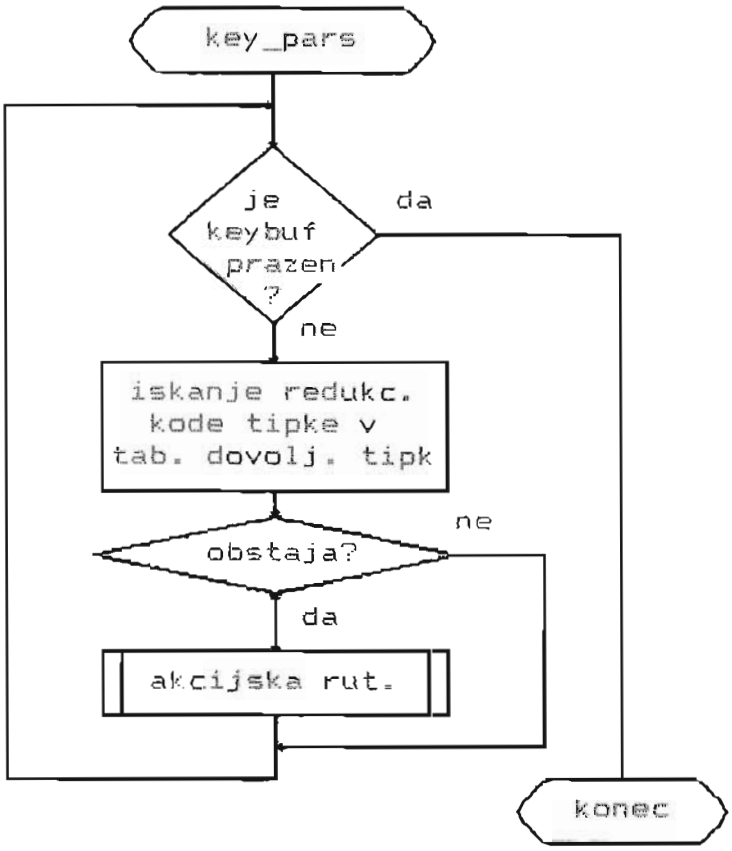
Na pritisnjeno tipko v nekem stanju sistema reagira rutina **key_pars**. Redukcijsko kodo tipke poišče v tabeli dovoljenih tipk tekočega stanja. Kazalec tabele dobi na state_stack-u preko curr_state kazalca. Če tipko najde, izvrši njeno akcijsko rutino. Za primer poglejmo obdelavo tipke v podrežimu Program list ročnega režima:



key_pars:

Nahaja se v osnovni zanki statex-a. Dokler vmesnik STATEX tipk in njihovih red. kod (**keybuf**) ni prazen, jemlje iz njega tipke. Redukcijsko kodo tipke primerja s kodami v tabeli dovoljenih tipk. Če jo najde (tipka je dovoljena), izvrši pripadajočo akcijsko rutino.

Vhod: keybuf

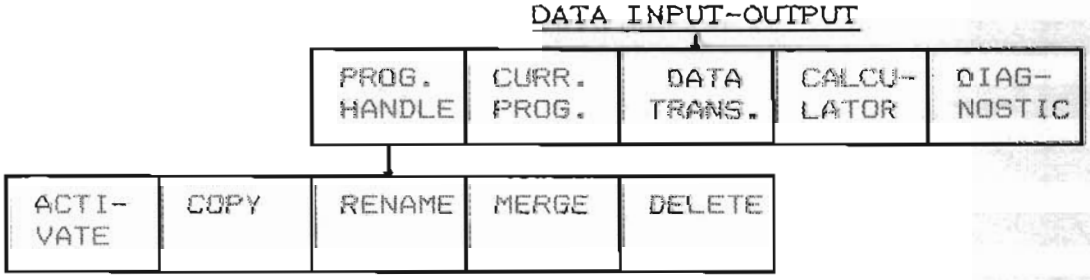


PREHAJANJE MED STANJI SISTEMA:

Možni so štirje načini prehajanja stanj:

1). s pritiskom na tipke glavnih režimov prehajamo med glavnimi režimi sistema (data in-out, manual, block by block in automatic normal). Če v nekem podrežimu (npr. automatic, program list) pritisnemo tipko tekočega glavnega režima (v tem primeru automatic) pridemo na osnovni nivo režima,

2). v podrežim nekega režima lahko pridemo s pritiskom ustrezne softkey tipke. Vsaka softkey tipka lahko predstavlja podrežim, kar tvori drevesno strukturo stanj. Za primer poglejmo eno od vej režima Data input-output.



3). prehod v podrežim je možen tudi v režimih s Parent Choose dialogom (glej poglavje 7),

4). iz podrežimov se vračamo na predhdne nivoje s pomočjo tipke /Đ .

Preklop glavnih režimov

Tipke glavnih režimov spadajo med STATEX tipke in se obdelujejo preko key_pars rutine. Prehodi med glavnimi stanji (oz. glavnimi režimi) niso vedno dovoljeni. Npr. med avtomatskim izvajanjem NC programa je prehod v nek drug glavni režim dela dovoljen šele po končanem programu ali pa po ustavitvi stroja s stop gobo. V tem primeru aktiviramo periodično rutino **wait_sw**, ki počaka na dovolitev preklopa.

Pri preklopu glavnih stanj potrebujemo naslednje spremenljivke:

- **ECMODE**, pove ali je preklop gl.stanja dovoljen (1-ni dovoljen),
- **MODE**, vsebuje številko aktivnega stanja,
- **dmode**, vsebuje številko željenega stanja,

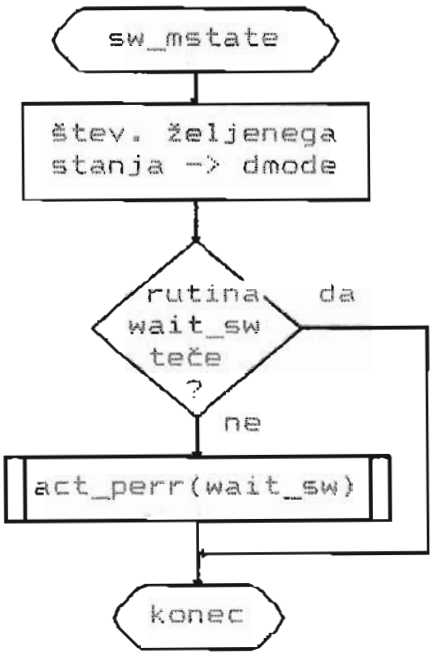
Poleg teh pa še:

- **ini_ruts**, tabela kazalcev na inic. rutine glavnih režimov,
- **waitsw_indx**, index periodične rutine wait_sw.

sw_mstate:

To je akcijska rutina tipk glavnih režimov. Glede na pritisnjeno tipko glavnega režima zapiše ustrezno številko v dmode. Če periodična rutina **waitsw** še ni aktivirana, jo aktivira.

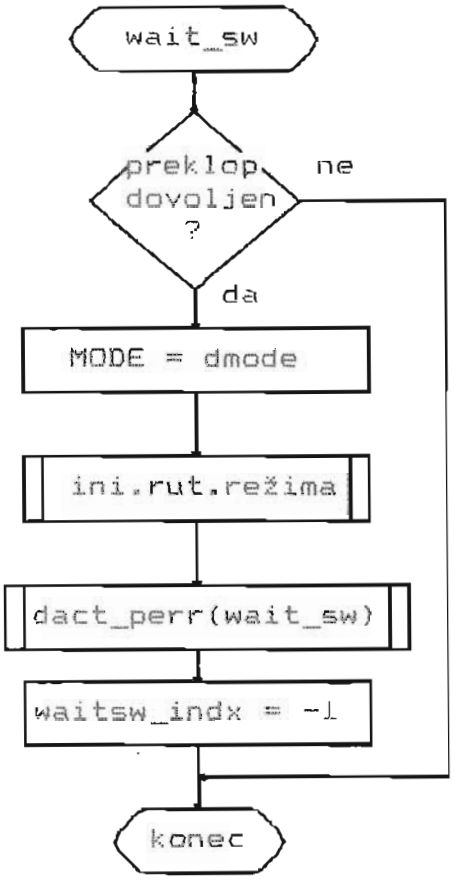
Vhod: key, koda pritisnjene tipke gl. režima
Sprememba:dmode, waitsw_indx
Rutine: wait_sw



wait_sw:

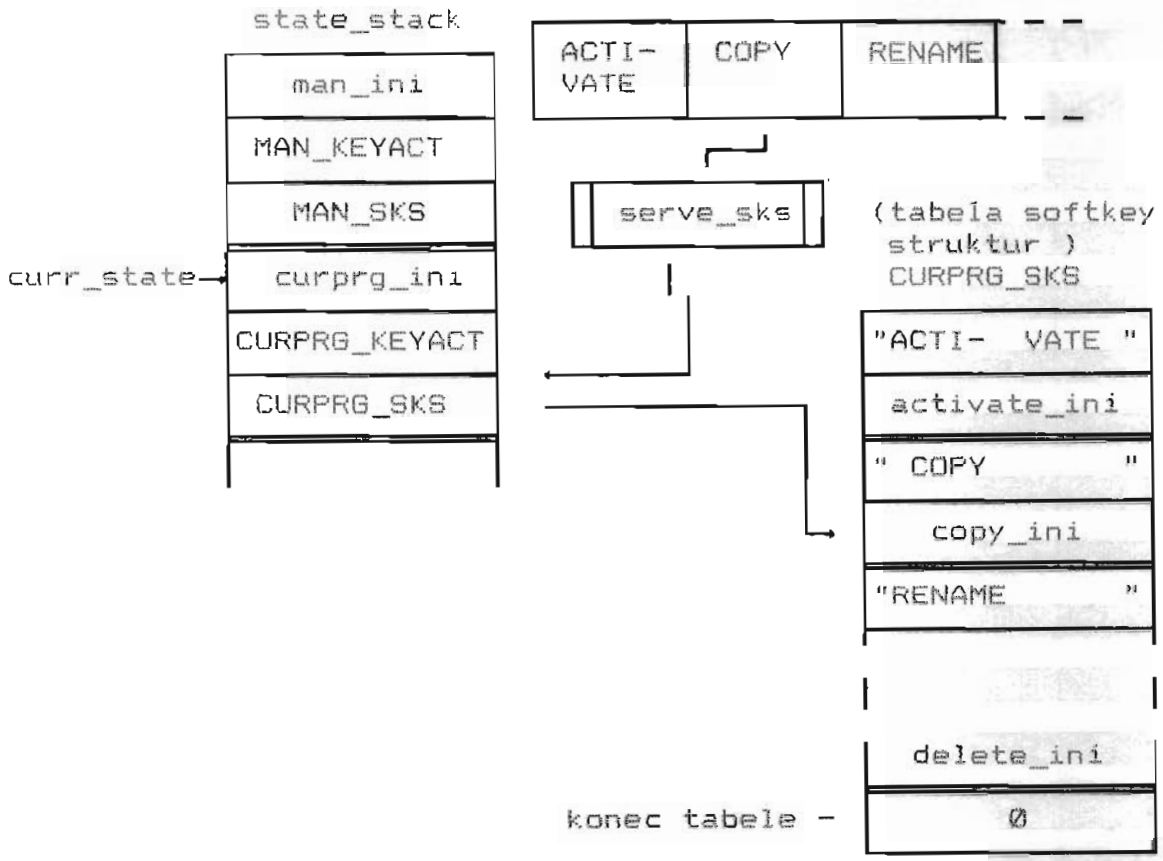
Je permanentna periodična rutina. Če je preklop dovoljen, vpiše številko novega stanja v MODE, izvede inicializacijsko rutino novega stanja ter se deaktivira. Če preklop ni dovoljen gre ven.

Sprememba:dmode, stevilka željenega stanja,
ECMODE, flag dovoljenosti preklopa,



Prehod v podrežim preko softkey tipk:

V podrežime ponavadi prehajamo preko softkey tipk. Rutina **serve_sks** pogleda ali je pritisnjena softkey tipka polna. Če je, izvede ustrezno rutino, ki je v primeru podrežimske softkey tipke inicializacijska rutino željenega podrežima.
Za primer poglejmo prehod v podrežim Rename Program list režima:



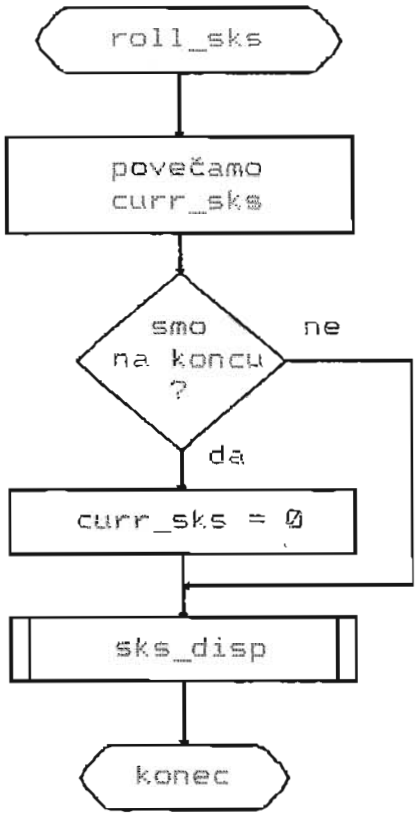
serve_sks:

Omogoča prehode v podstanja glavnega režima ali pa dialog (odvisno od softkey rutine). Aktivira jo key_pars rutina ob pritisku na eno od softkey tipk. Rutina serve_sks pogleda katera softkey tipka je bila pritisnjena. Če je tipka polna, izvede ustrezno rutino softkey rutino.

roll_sks:

Je akcijska rutina tipke za izbiro strani softkey tipk (>) v režimih, ki imajo več softkey strani. Izpiše naslednjih pet softkey tipk režima, če pa smo na koncu softkey tabele, izpiše prvih pet (krožno). Ustrezno spremeni curr_sks števec.

Vhod: curr_sks
Rutine: sks_disp - izpiše softkey nize



Vrnitev iz podrežima:

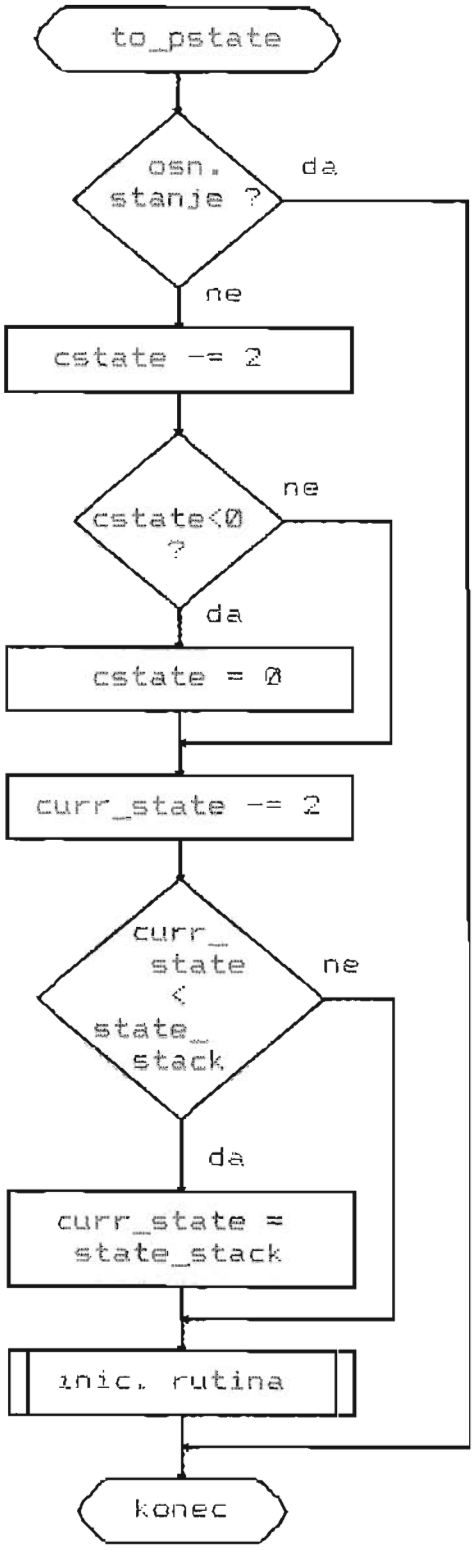
V predhodno stanje se vrnemo s pritiskom na tipko /N, ki aktivira rutino **to_pstate**. Ta dekrementira curr_state in **cstate** (kazalec na številko tekočega stanja v vmesniku številk prehojenih stanj **state_buf**) za 2 in zažene inicializacijsko rutino predhodnega stanja.

state_buff služi za shranjevanje številk stanj skozi katere smo šli od glavnega režima do tekočega podrežima. Indeks **cstate** kaže vedno na številko tekočega stanja. S pomočjo teh dveh podatkovnih struktur imamo zelo enostaven in hiter dostop do številke tekočega stanja, številk predhodnih stanj in globine tekočega stanja, ki ji potrebujemo pri izpisovanju imen režimov in podrežimov na ekran.

to_pstate :

Ob pritisku tipke za prehod v predhodno stanje /N dekrementira curr_state in cstate tako, da kažeta na predpredhodno stanje oz. na osnovno stanje na state_stack-u in v statebuf-u. Nato izvede inicializacijsko rutino predhodnega stanja. Če smo ob

pritisku /R že v osnovnem stanju ni nobene akcije.
Sprememba: cstate, curr_state.



Inicializacijske rutine režimov in podrežimov:

Razlika med inic. rutinami glavnih režimov in podrežimov je v tem, da inic. rutine glavnih režimov zapišejo svoje podatke na začetek state_stack-a in inicializirajo kazalec curr_state, inic. rutine podrežimov pa inkrementirajo curr_state, da kaže na prazno mesto state_stack-a in tja zapišejo podatke. Podobno je z zapisom številke tekočega stanja v vmesnik številk stanj **state_buf**.

Vse inicializacijske rutine pa storijo še naslednje:
- inicializirajo ekran,
- izpišejo napise softkey tipk,
- izpišejo ime stanja (režim ali podrežim),
- opravijo osnovne operacije potrebne za delovanje režima oz. podrežima.

Podatkovne strukture avtomata:

state_stack: področje rezervirano za shranjevanje podatkov stanj . Velikost je n * 3 * 4 (byte-ov), kjer je n število stanj najglobjega režima sistema,

curr_state: kazalec na podatke tekočega stanja na state_stack-u (long),

state_buf: vmesnik za shranjevanje številk stanj skozi katere smo šli od glavnega režima do tekočega podrežima. Velikost state_buf-a je n byte-ov, kjer je n število stanj najglobjega režima sistema,

cstate: word-ni offset številke tekočega stanja od začetka state_buf-a,

***_KEYACT:** tabele dovoljenih tipk in akcijskih rutin vseh režimov in podrežimov sistema,

***_SKS:** tabele napisov softkey tipk in ustreznih rutin režimov in podrežimov sistema,

curr_fks: številka tekoče strani softkey-ev (0 = 1. stran),

dmode: byte-ni vmesnik za številko željenega glavnega režima,

ini_ruts: tabela kazalcev na inicializacijske rutine glavnih režimov (4 longi),

waitsw_indx: word-ni vmesnik za shranitev indeksa periodične rutine wait_sw,

2. P E R I O D I Č N E R U T I N E

Poleg procesiranja dogodkov (obdelave tipk) lahko v STATEX-u tudi spremljamo procese. To nam omogočajo periodične rutine. Če želimo, da se nam neka rutina izvaja periodično, samo pokličemo sistemsko rutino **act_perr**, ki uvrsti njen kazalec v tabelo periodičnih rutin **perr_area**. Druga sistemska rutina **perex** nato skrbi, da se ta rutina izvede ob vsakem obhodu glavne STATEX zanke (vsakih 30-40 ms).

Vsaki periodični rutini v tabeli perr_area pripada poleg njenega kazalca (long) še tip (word), ki pove :

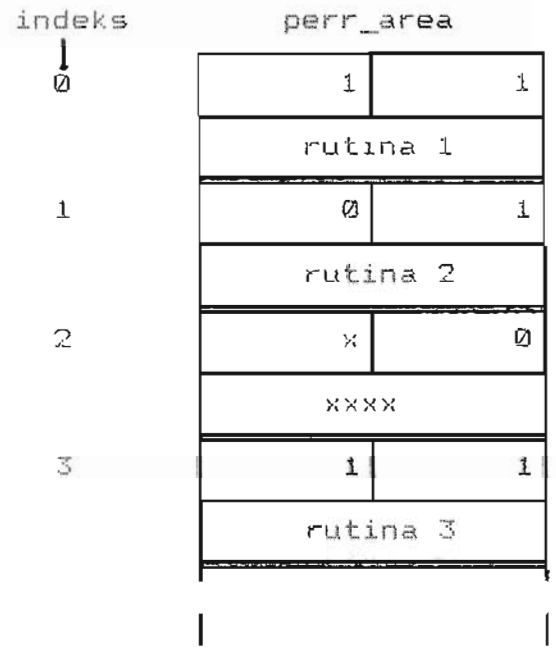
- ali je kazalec veljaven, t.j. ali je to mesto tabele polno (lsbyte = 1),
- ali je rutina permanentna (msbyte = 1),

Byte veljavnosti naslova nam omogoča hitrejše testiranje polnih mest v tabeli perr_area.

Byte permanentnosti omogoča obstoj periodičnih rutin, ki jih lahko deaktivira le modul, ki jih je aktiviral in so tako zaščitene pred drugimi moduli.

V tabeli periodičnih rutin je prostor za deset periodičnih rutin.

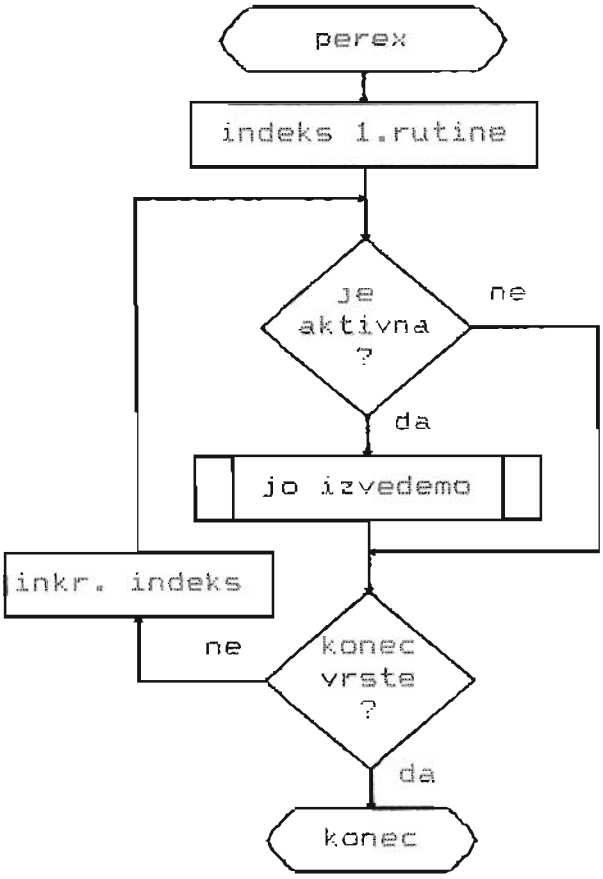
Za primer poglejmo del perr_area tabele v nekem trenutku:



SISTEMSKE RUTINE ZA OBDELAVO PER. RUTIN :

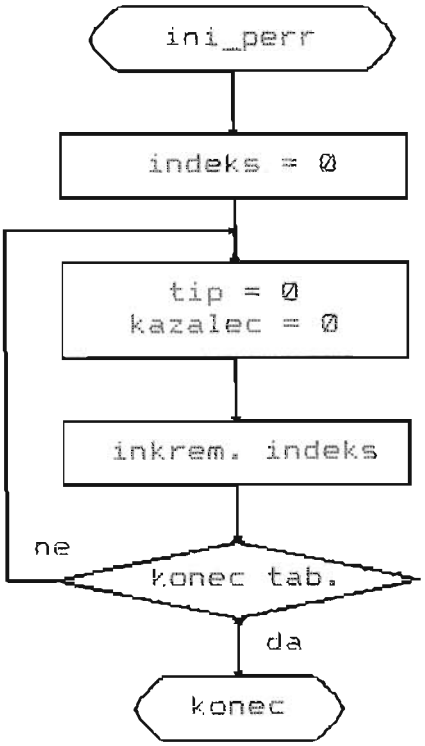
perex:

Izvede vse aktivne per. rutine v tabeli perr_area. Nahaja se v glavni zanki STATEX-a in se izvede ob vsakem njenem obhodu.



ini_perr:

Inicializira tabelo per. rutin perr_area. Je del splošne inicializacije STATEX modula.

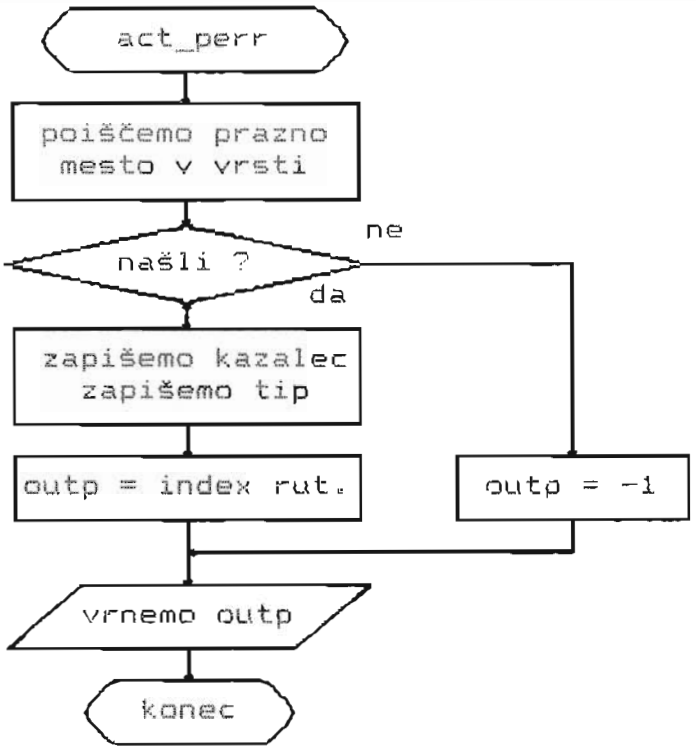


act_perr:

Uvrsti rutino v tabelo perr_area in jo aktivira.

Vhod: kazalec na rutino, tip rutine

Izhod: - index rutine (OK), -1 = napaka



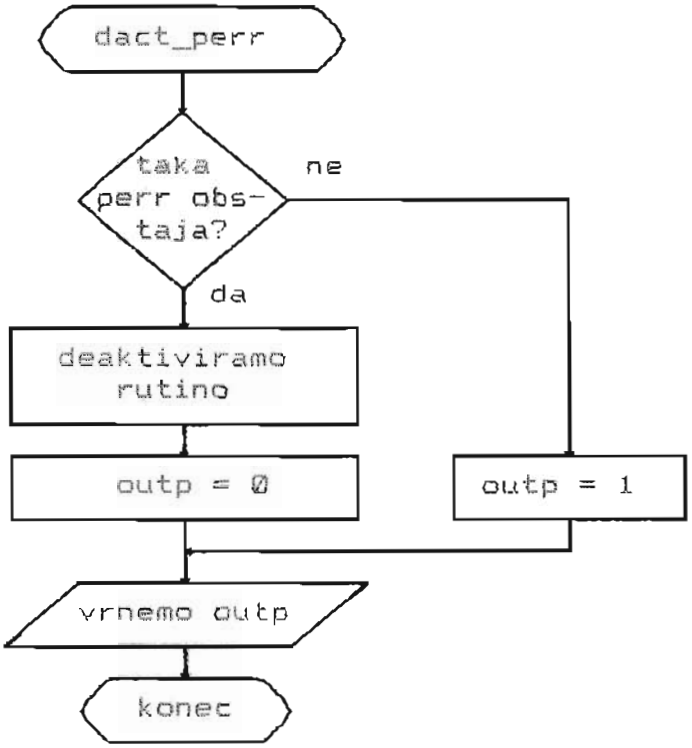
dact_perr:

Deaktivira perr. rutino z indeksom index.

Vhod: int index,

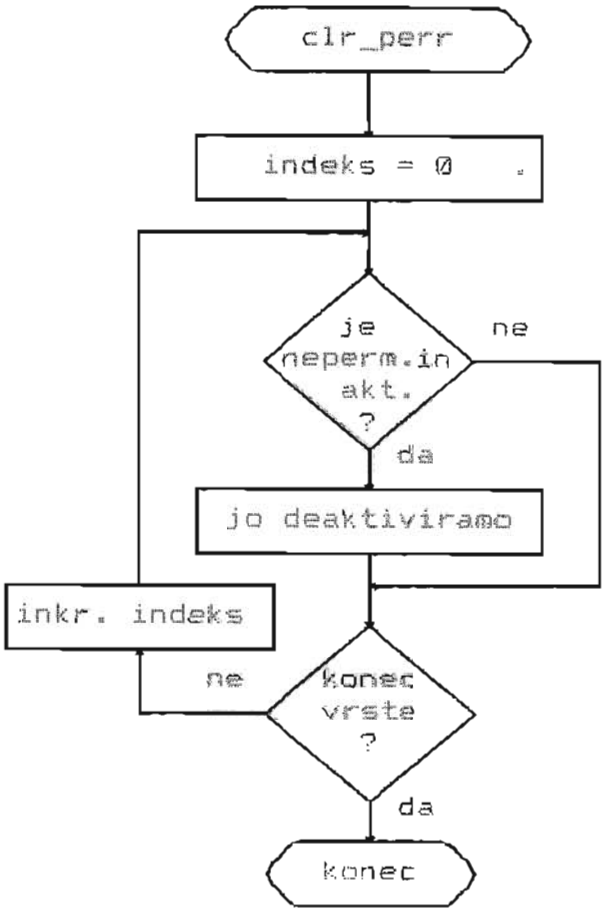
Izhod: - 0, rutina deaktivirana

- ≠0, ni perr. rutine s takim indeksom.



clr_perr:

Deaktivira vse nepermanentne periodične rutine v tabeli perr_area.

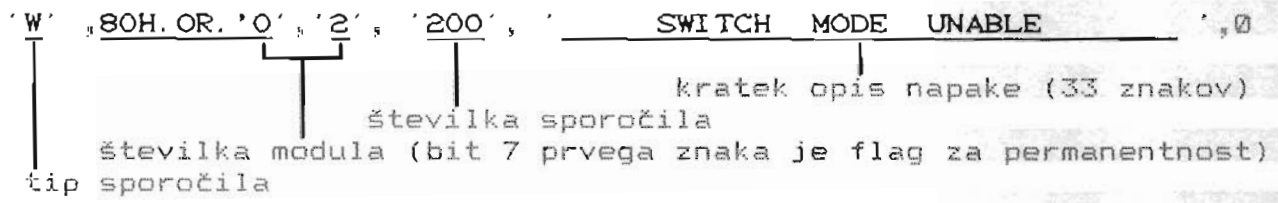


Podatkovne strukture periodičnih rutin:

perr_area: tabela periodičnih rutin velikosti 10 * (2+4) bytov

3. D I A G N O S T I K A

Sistem LJUMO pozna tri vrste sporočil, preko katerih moduli sistema seznanjajo operaterja o dogodkih na stroju, napakah v uporabniških programih itd. Vrste sporočil so: alarm-i, warning-i in message-i. Ločijo se po prvem ASCII znaku v sporočilu, ki je enak prvi črki vrste sporočila (A-alarm, W-warning, M-message). Poglejmo si obliko sporočila:

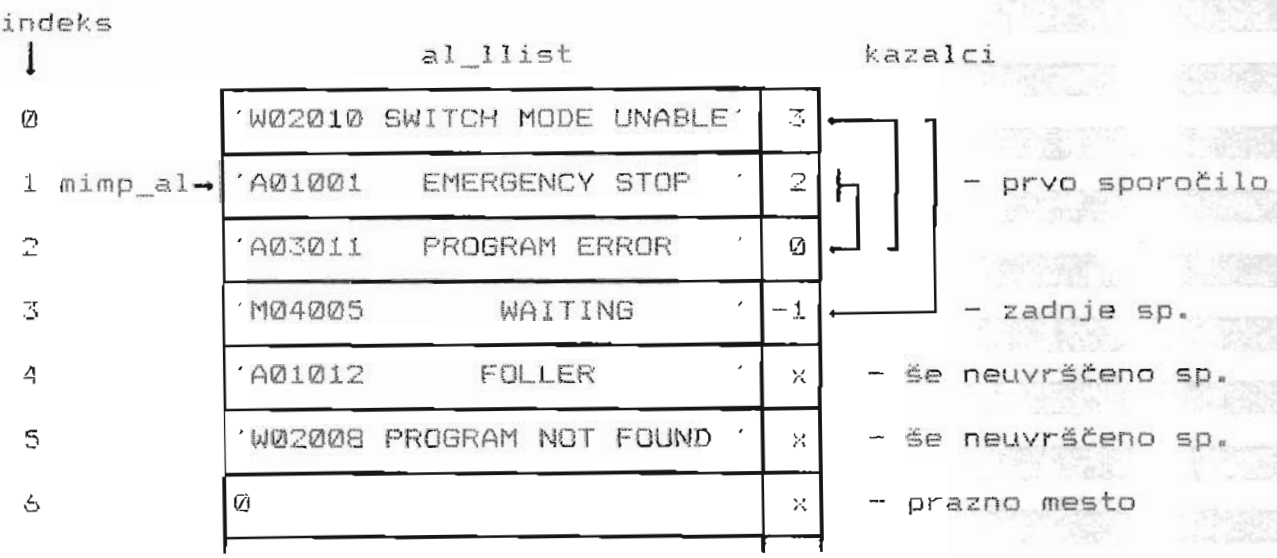


Poleg tega se sporočila ločijo na nepermanentna in permanentna. Permanentna sporočila imajo postavljen bit 7 prvega znaka številke modula, pri izpisu sporočil v podrežimu DIAGNOSTIC pa so označeni z zvezdico. Nepermanentna sporočila lahko brišemo s pritiskom na tipko DIAG v stanju DIAGNOSTIC, permanentna pa lahko brišejo le moduli, ki so jih postavili.

Prvih 6 znakov sporočila služi za določitev njegove prioritete. Pravila določitve prioritete so :

1. alarm > warning > message,
2. permanentna > nepermanentna,
3. moduli z manjšim stevilom > moduli z večjim > PC,
4. sporočila z nižjim številom > sporočila z višjim številom.

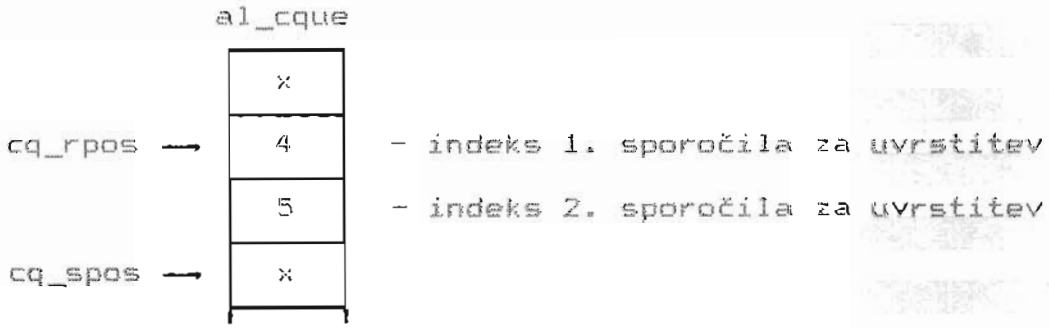
Glavna podatkovna struktura diagnostike je tabela **al_llist** (alarm linked list) v kateri so sporočila urejena po prioriteti. Organizirana je kot Singly linked list kar pomeni, da vsakemu sporočilu v tabeli pripada poleg stringa sporočila tudi indeks naslednjega spročila. Zadnje sporočilo v vrsti ima indeks -1. Indeks prvega sporočila vsebuje globalna spremenljivka **mimp_al** (most important alarm). Prazno mesto v vrsti prepoznamo po praznem alarmnem stringu (le znak Ø). V tabeli je prostor za 32 sporočil. Poglejmo si primer al_llist tabele:



Poleg tega obstaja še dodatna podatkovna struktura **al_cque** (alarm circular queue). To je krožni vmesnik za word-ne indekse sporočil, ki čakajo na uvrstitev. Omogoča nemoteno uvrščanje sporočil različnih taskov. Za delovanje krožne vrste potrebujemo dva indeksa:

- **cq_spos** (circular queue store position), kaže na prazno mesto za vpis indeksa novega sporočila,
- **cq_rpos** (circular queue retreive position), kaže na sporočilo ki je na vrsti za uvrstitev.

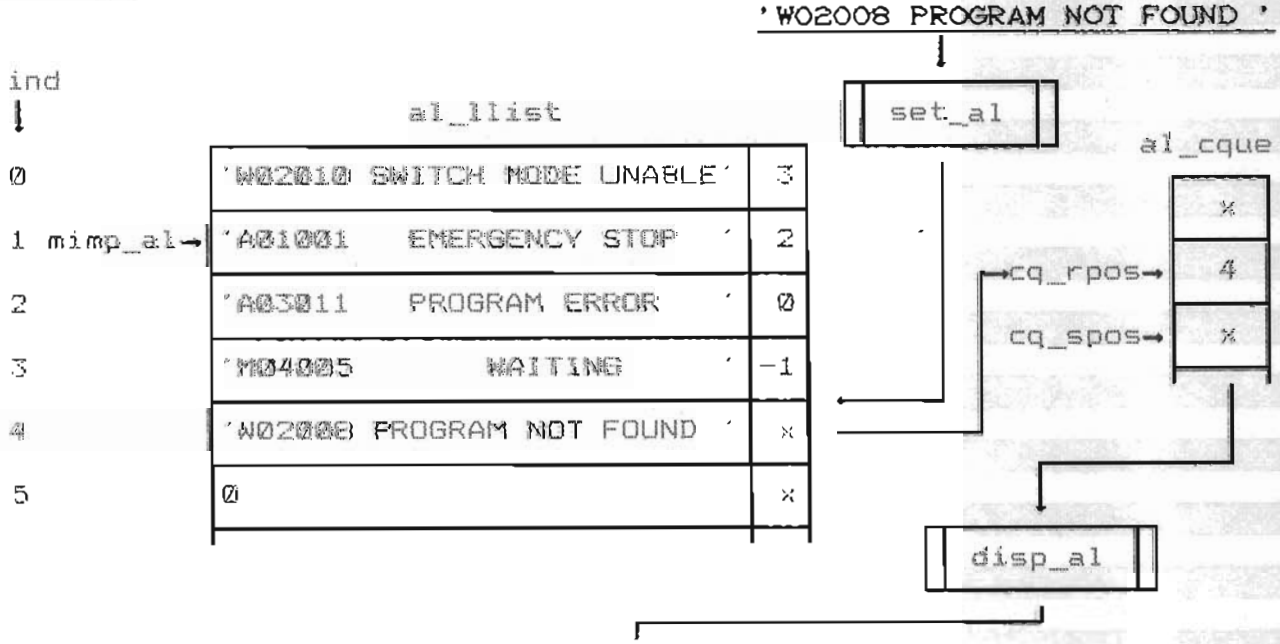
Ti spremenljivki nam povesta tudi, kdaj je vmesnik prazen (cq_spos = cq_rpos) in kdaj je polen (cq_spos+1 = cq_rpos ali cq_spos+1 = MAX in cq_rpos = 0). V vrsti je prostor za 10 indeksov čakajočih rutin. Poglejmo si al_cque vrsto za prejšnji primer:



Osnovni princip uvrščanja sporočil:

Modul, ki želi oddati sporočilo, pokliče sistemsko rutino **set_al**, kateri posreduje kazalec na sporočilo. Rutina prepiše sporočilo na prvo prazno mesto v vrsti urejenih sporočil al_llist, njen indeks pa zapiše v krožno vrsto indeksov čakajočih sporočil al_cque ter ga vrne uporabniku. Ob vsakem obhodu glavne zanke STATEX-a se izvede rutina **disp_al**, ki nove čakajoče alarme uvrsti v al_llist na mesta, ki jim po prioriteti pripadajo (uredi indekse sporočil v vrsti) in skrbi za izpis najpomembnejšega sporočila v diagnostični vrsti (druga vrsta ekrana).

Primer uvrstitve sporočila:



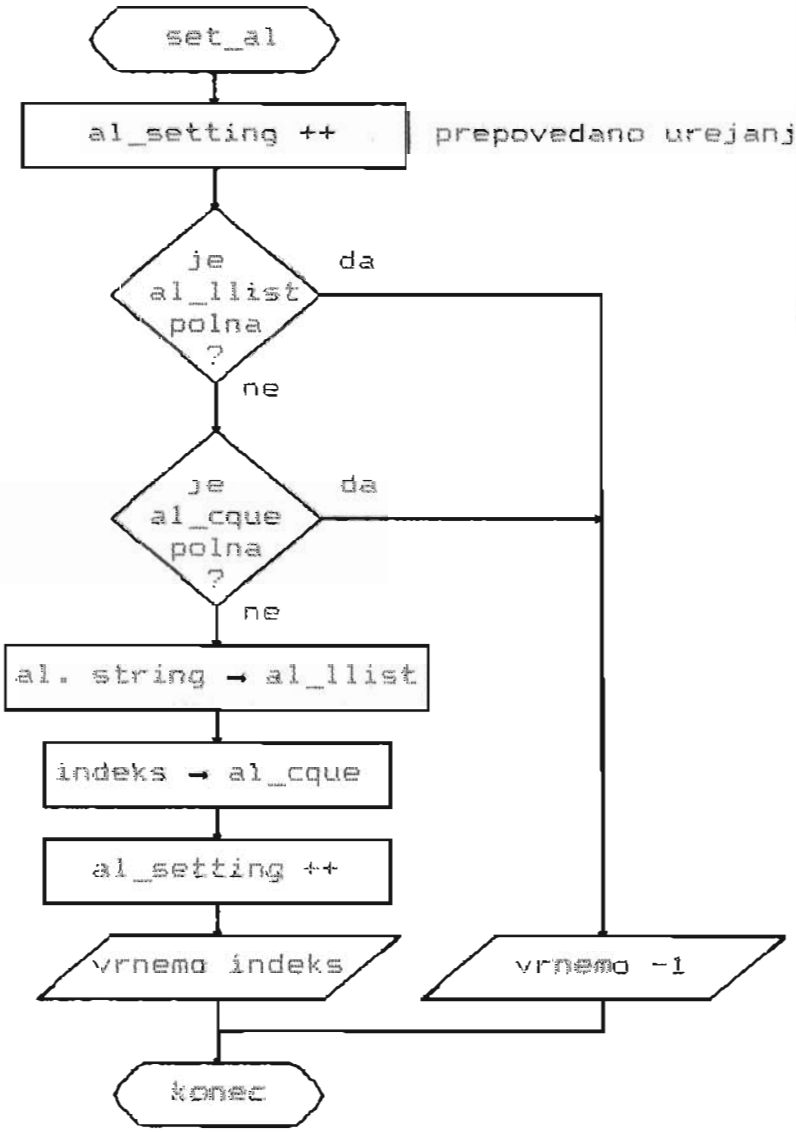
ind	al_llist		
0	'W02010 SWITCH MODE UNABLE'	3	
1 mimp_al→	'A01001 EMERGENCY STOP '	2	
2	'A03011 PROGRAM ERROR '	4	! sprememba
3	'M04005 WAITING '	-1	
4	'W02008 PROGRAM NOT FOUND '	0	! je uvrščen

DIAGNOSTIČNE RUTINE:

set_al:

Prepiše sporočilo v tabelo al_llist na prvo prazno mesto, njegov indeks pa uvrsti v vmesnik čakajočih sporočil al_cque.

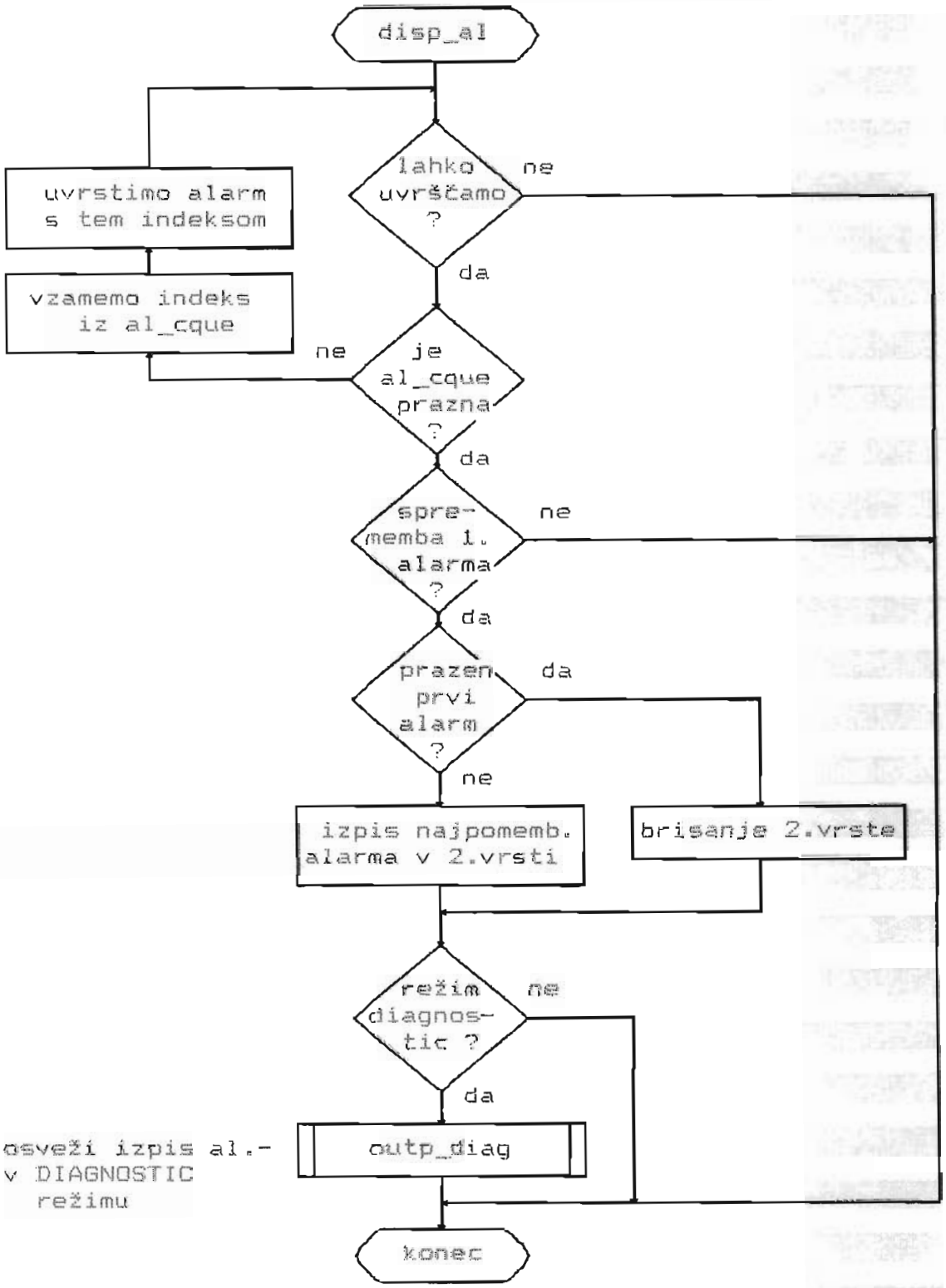
Vhod: - kazalec na sporočilo,
Izhod: - OK-indeks sporočila, -1 če ga ni uvrstil



disp_al:

Če je dovoljeno urejanje al_llist-a (al_setting = 0), pogleda v al_cque in če ni prazen gre na urejanje. Če je prišlo do spremembe najpomembnejšega sporočila pogleda, ali je polno. Če je, izpiše najpomembnejše sporočilo v alarmni vrsti, če pa je prazno, alarmno vrstico pobriše. Ta rutina se izvede ob vsakem obhodu glavne STATEX zanke.

Rutine: - bigger_al, blink_mess

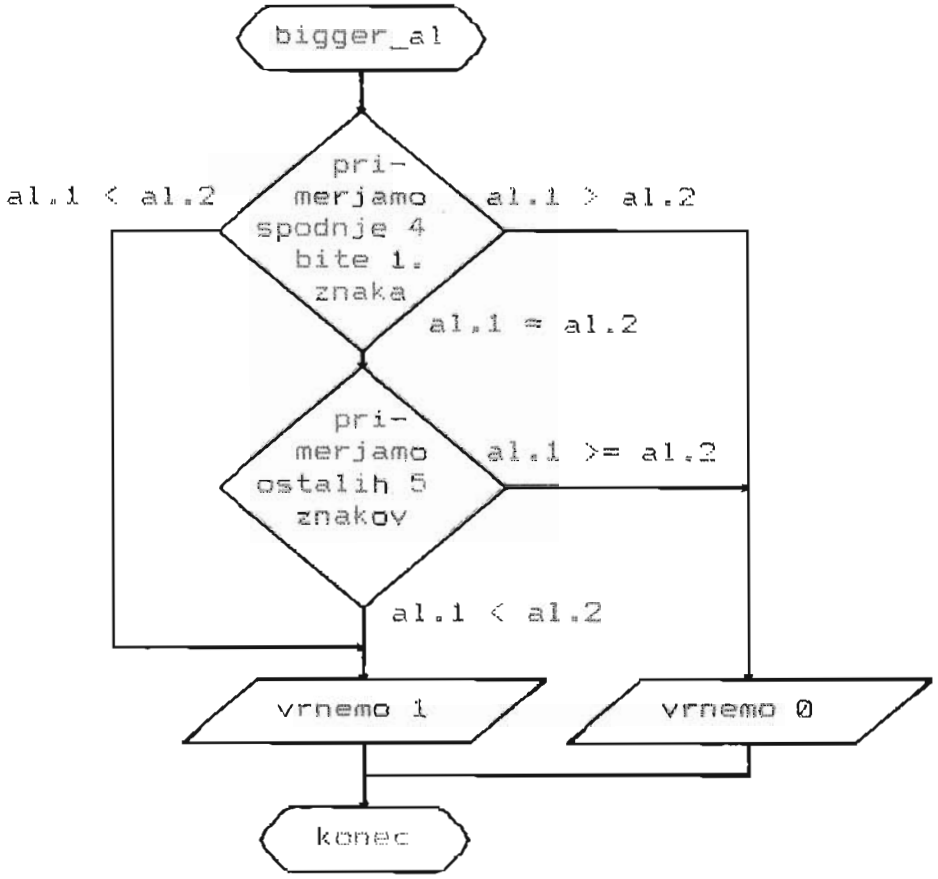


osveži izpis al.- v DIAGNOSTIC režimu

bigger_al:

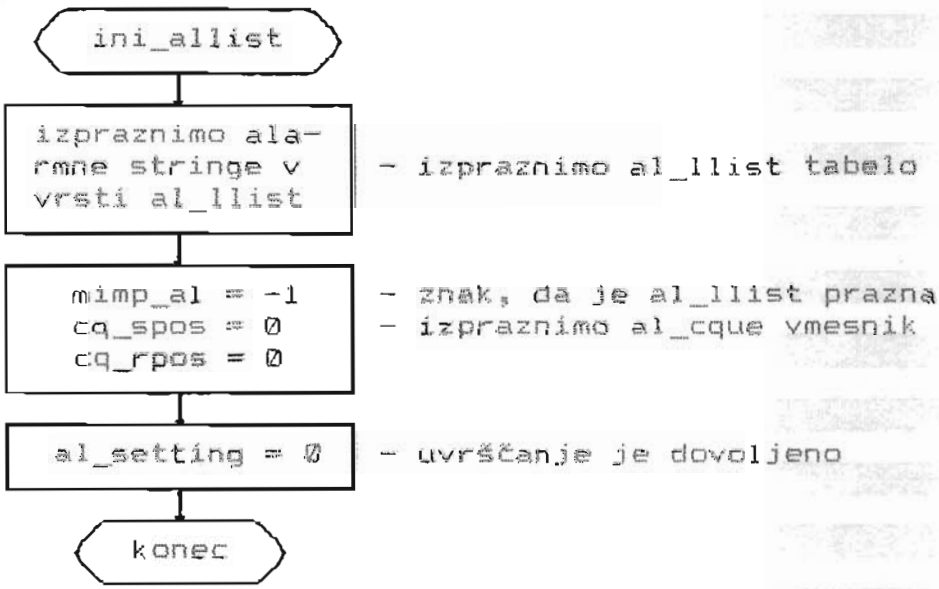
Ugotovi ali ima prvi alarm višjo prioriteto od drugega. Princip primerjave je ta, da primerjamo le nižje štiri bite prvega znaka (ker A=41H, W=57H in M=4DH, nižje število pa pomeni višjo prioriteto). Ostalih pet znakov primerjamo normalno.

Vhod: kazalec na 1. alarm in kazalec na drugega,
Izhod: - 1, prvi je večji; 0, prvi je manjši ali enak.



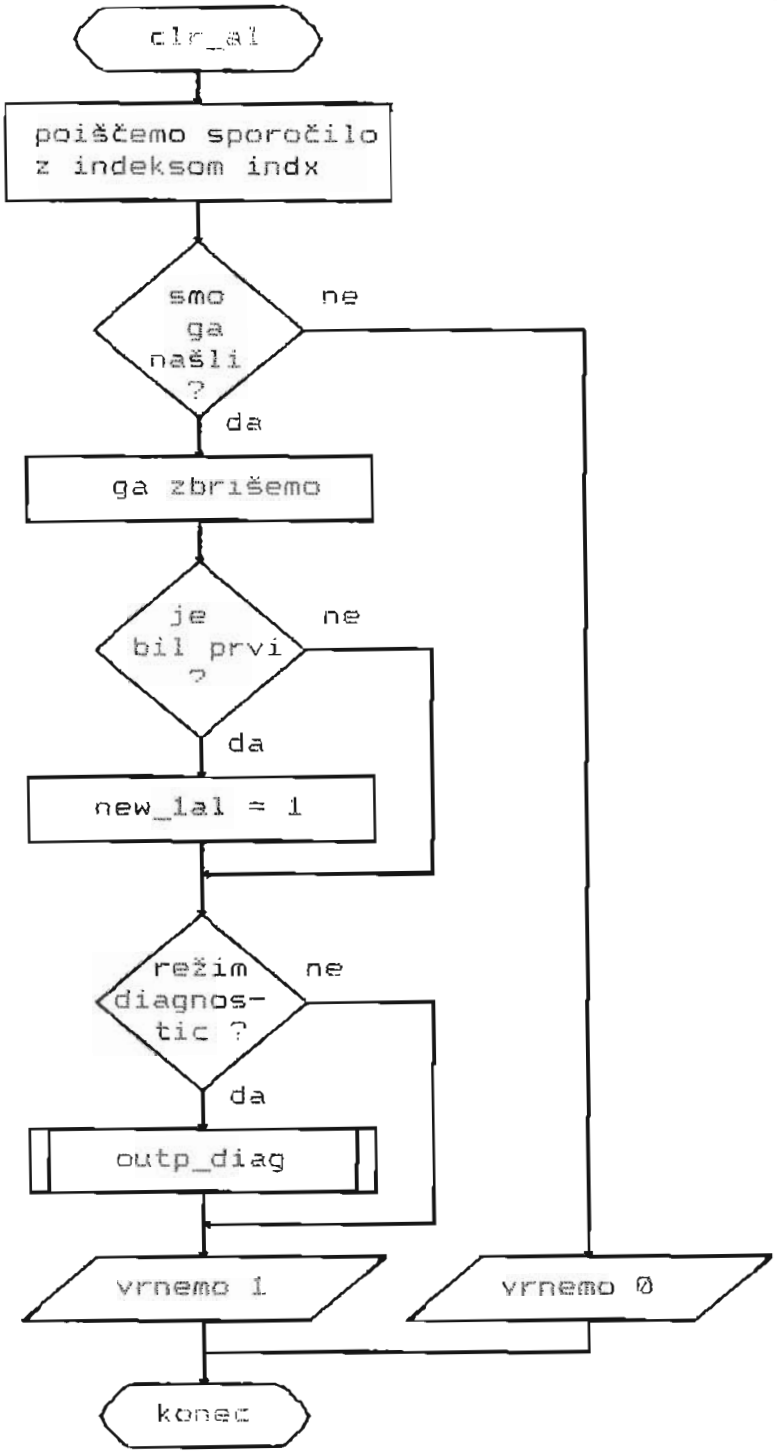
ini_allist:

Inicializira vrsto sporočil al_llist in krožni vmesnik al_cque.



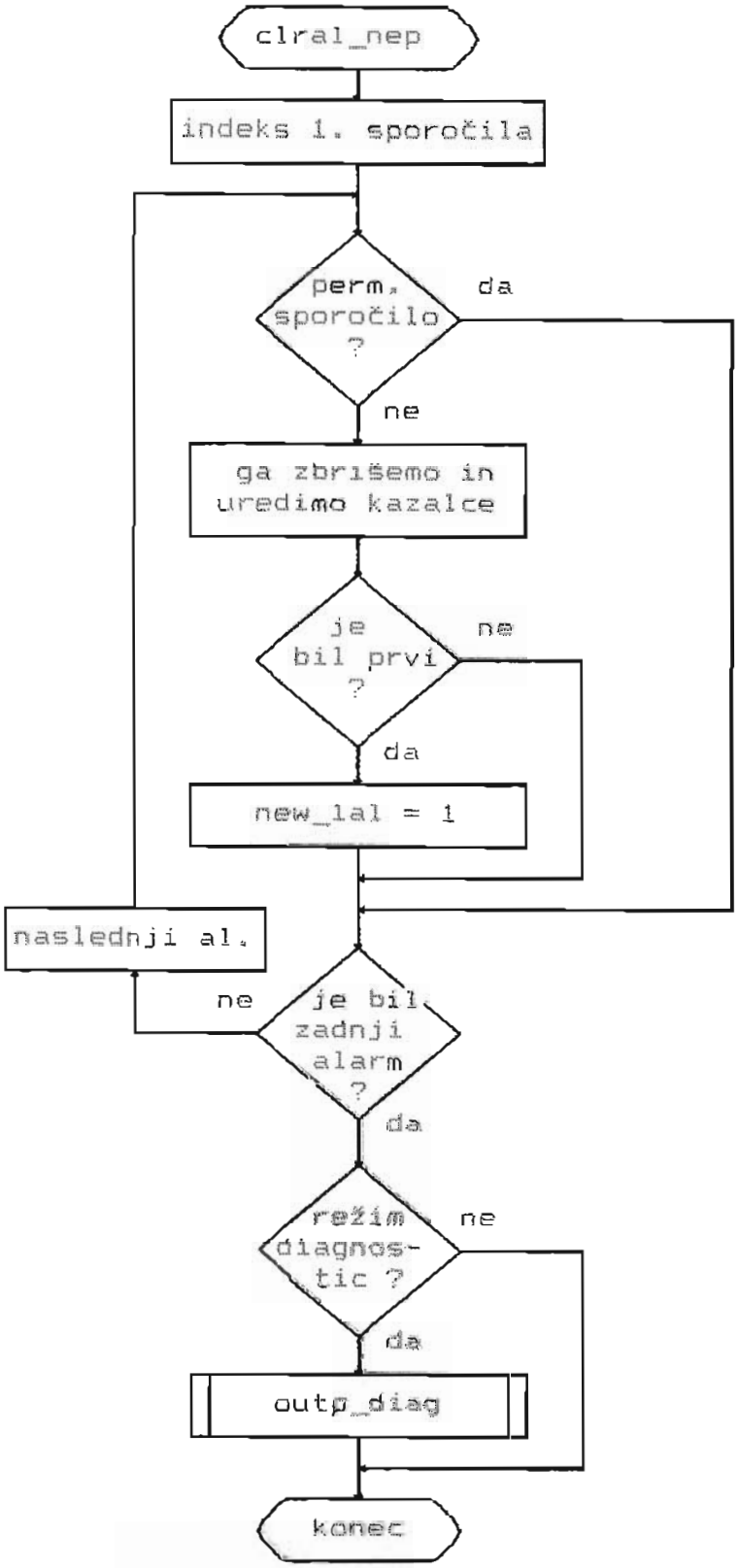
clr_al:

Briše sporočilo z indeksom indx.
Vhod: indx sporočila,
Sprememba: mimp_al, če je zbrisan prvi alarm,
Izhod: - 0, sporočilo je zbrisano,
- 1, ni sporočila s takšnim indeksom v vrsti.



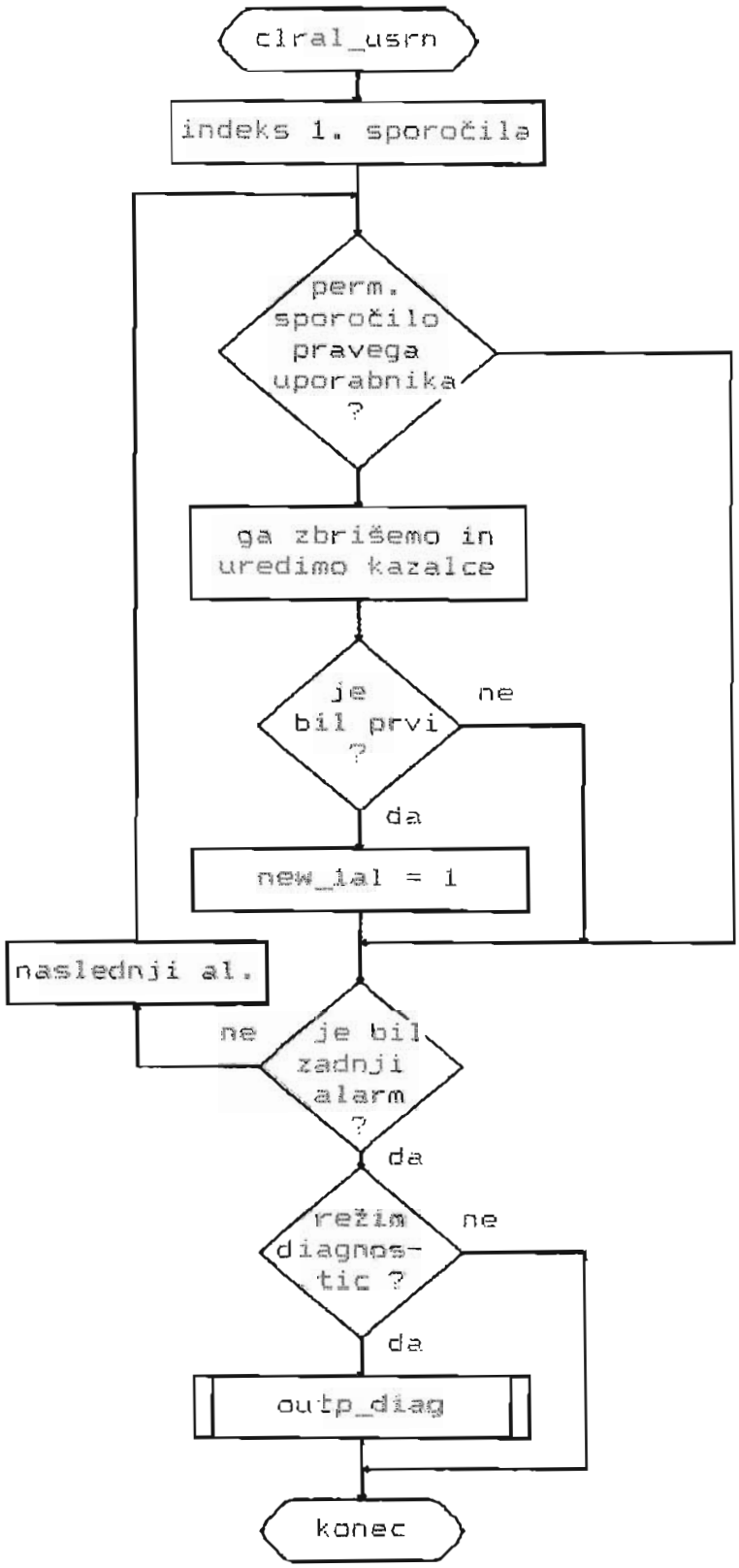
clral_nep:

Briše vsa nepermanentna sporočila iz vrste sporočil.
Sprememba: mimp_al, če je zbrisan prvi alarm.



clral usrn:

Briše vsa nepermanentna sporočila user_numb uporabnika.
Vhod: številka modula,
Izhod: 0 - zbrisano vsaj eno sp.; 1 - ni neperm. sp.

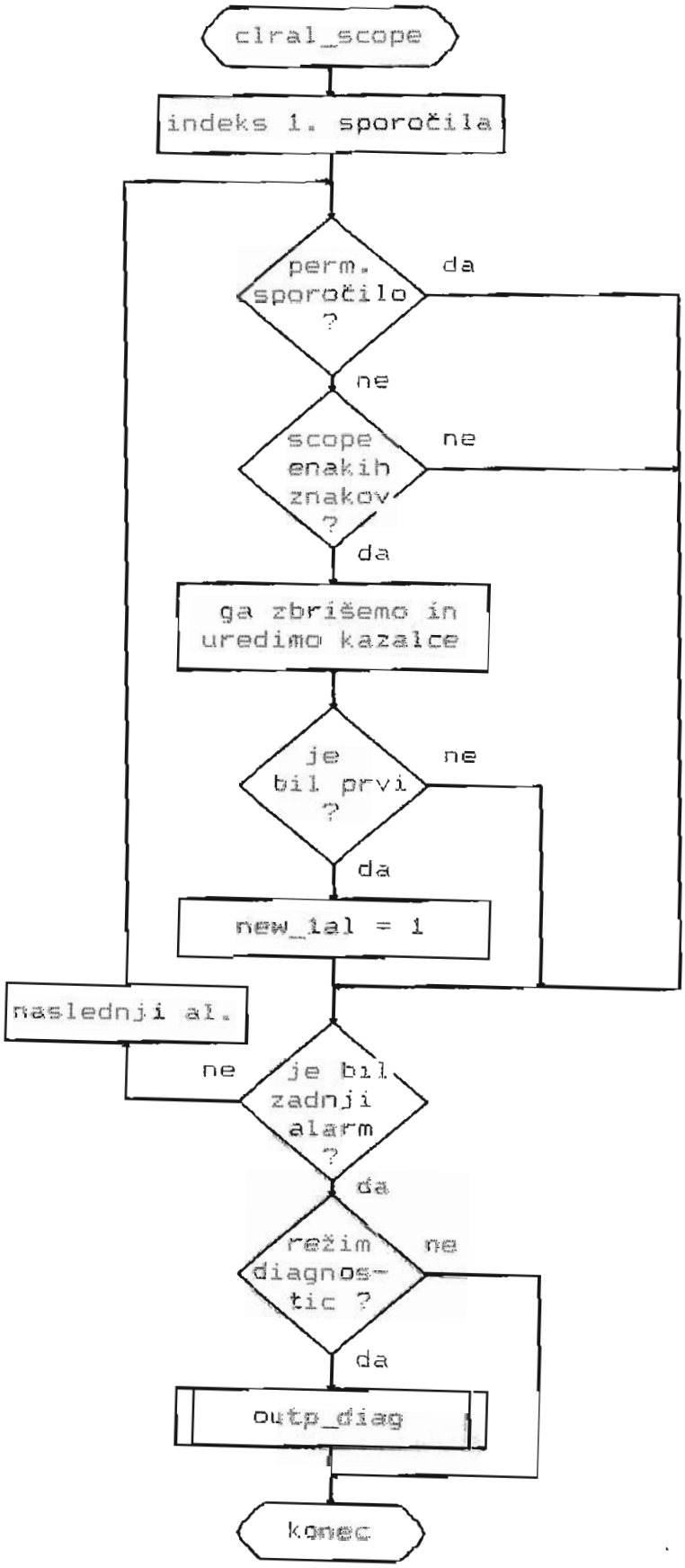


clral scope:

Zbriše vsa tista nepermanentna sporočila uporabnika ,ki imajo scope enakih znakov za prvim znakom sporočila (A,W,M). Število scope mora biti najmanj 2, da lahko prepoznamo uporabnika.

Vhod: strn, kazalec na string znakov za primerjavo,
scope, število znakov, ki naj se primerjajo,

Izhod: 0 - zbrisano vsaj 1 sp.; 1 - ni neperm. sp.



Podatkovne strukture diagnostike:

al_llist: urejena vrsta diagnostičnih sporočil. Velikost je 32*6bytov,

mimp_al: word-ni indeks alarma z najvišjo prioriteto,

new_1al: word-ni znak za spremembo prvega alarma,

al_setting: word, če je < 0 pove, da urejanje al_llista ni dovoljeno,

al_cque: krožni vmesnik indeksov sporočil, ki čakajo na uvrstitev v al_llist. Velik je 10 byte-ov.

cq_spos: word-ni indeks prostega mesta za vpis v al_cque,

cq_rpos: word-ni indeks tekočega mesta za branje iz al_cque.

4. UREJANJE EDITORSKEGA VMESNIKA

Editorski vmesnik (**editbuf**) sistema LJUMO je namenjen uporabniku za urejanje in vnašanje raznih podatkov v sistem kot npr. NC programiranje, vnašanje vrednosti korekcij orodij, premikov ničelnih točk, izbiranje in manipulacija s programi itd. Dolžina editorskega vmesnika je 240 znakov.

Editorsko polje ekrana (16. in 17. vrstica) prikazuje izrez 78 znakov editorskega vmesnika. Vsaka sprememba besedila v tem delu editorskega vmesnika se ustrezno prikaže na ekranu. Za to preslikavo skrbi rutina **disp_eb**.

Mesto prvega znaka v 16. vrstici je rezervirano za znak <, ki pove, da se pred prvim znakom izpisanim v 16. vrsti še nahaja besedilo, ki pa ni vidno. Z editorsko tipko ← se lahko premaknemo na skrito besedilo. Ko pridemo na začetek 16 vrstice se prikaz v 16. in 17 vrstici pomika v levo (scroll right). Mesto zadnjega znaka v 17. vrstici je rezervirano za znak >, ki pove, da se za zadnjim znakom izpisanim v 17. vrsti še nahaja besedilo, ki prav tako ni vidno. Z editorsko tipko → se lahko premaknemo na skrito besedilo. Ko pridemo na konec 17 vrstice se prikaz v 16. in 17 vrstici pomika v desno (scroll left).

Za obdelavo teksta v editorskem vmesniku obstaja nekaj preprostih editorskih rutin.

Za obdelavo besedila v ed. vmesniku potrebujemo štiri spremeenljivke:

- **curchr**, števec odmika tekoče pozicije od začetka vmesnika,
- **lastchr**, števec odmika zadnjega znaka od začetka vmesnika.
- **first_seen**, odmik prvega prikazanega znaka od začetka editorskega vmesnika,
- **last_seen**, odmik zadnjega prikazanega znaka od začetka,

EDITORSKE RUTINE

clr_eb:

editorski vmesnik napolni s presledki do MAXEB (239), inicializira curchr, lastchr, first_seen in last_seen ter osveži prikaz v 16. in 17. vrsti.

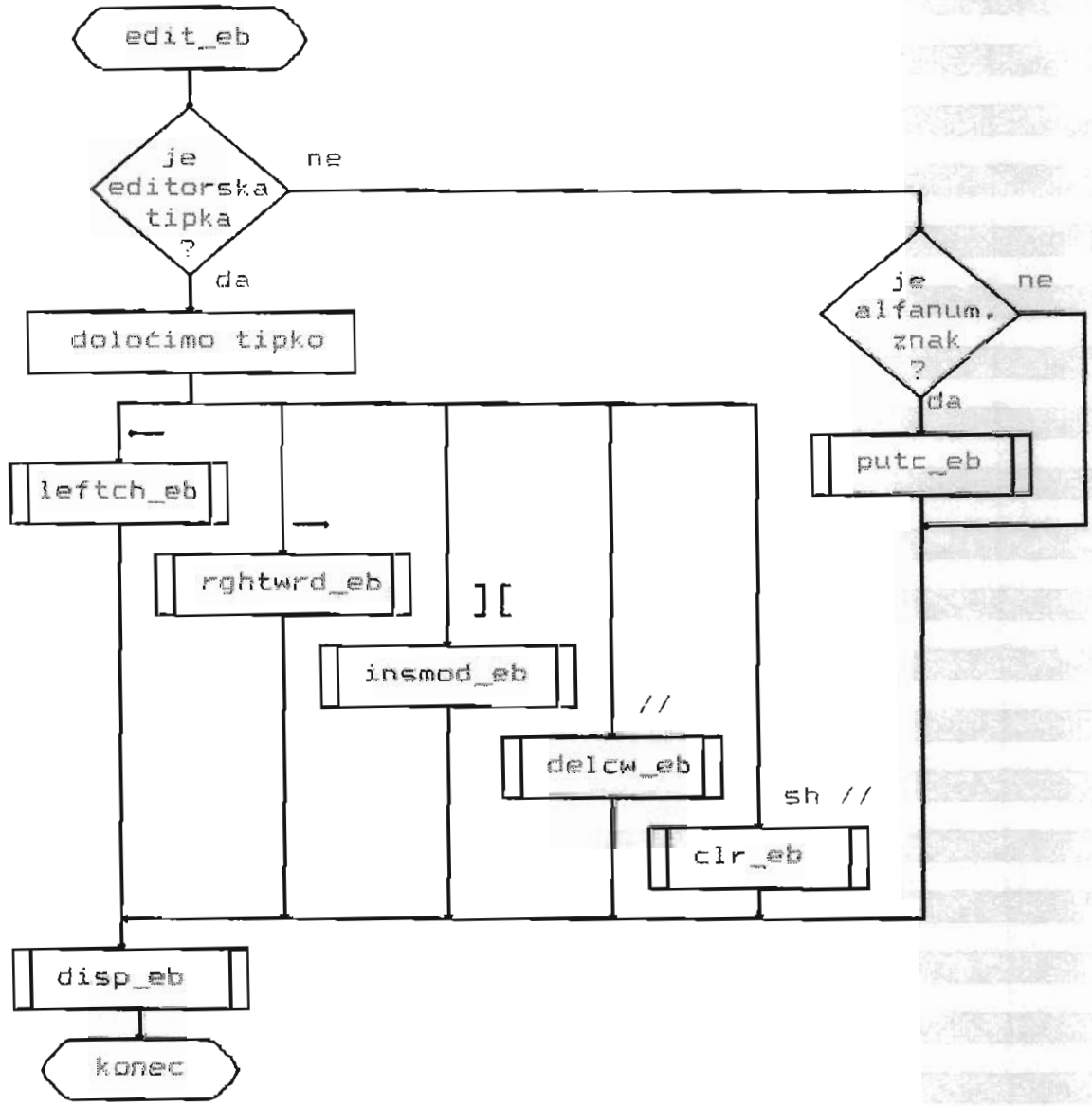
Sprememba: curr_chr, last_chr, first_seen, last_seen
Rutine: disp_eb

– prikaže ed. vmesnik na ekranu

edit_eb:

To je akcijska rutina, ki se izvede ob pritisku ene od editorskih tipk (v stanjih, ki omogočjo editiranje), alfanumeričnih tipk ali tipk za ločila. Vsebuje glavno editorsko zanko, ki glede na tip pritisnjene tipke izvrši ustrezno editorsko funkcijo.

Vhod: rkeyb, keyb,
Rutine: disp_eb, editorske rutine,

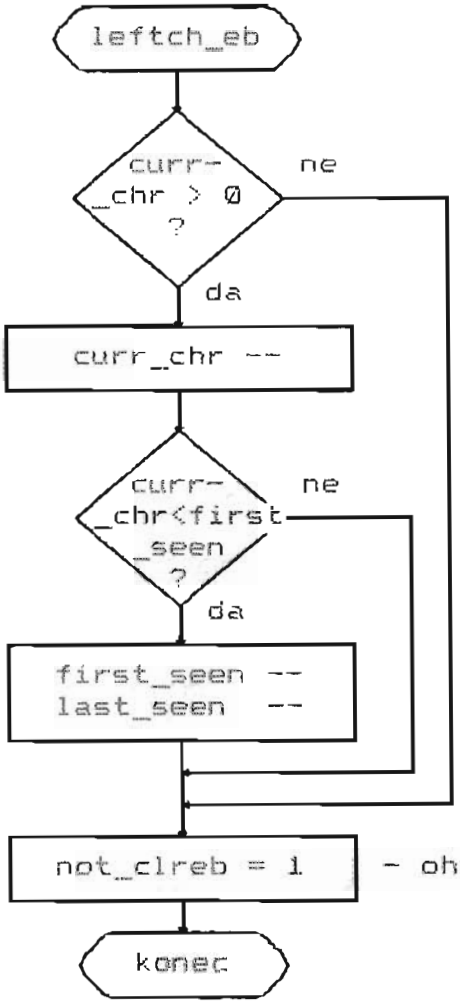


leftch_eb:

Pomik kursorja za en znak v levo. Če smo na začetku editbuf-a ni akcije. Če smo na začetku 16. vrstice, ne pa tudi na začetku editbuf-a, se ekran scroll-ira v desno.

Sprememba: curr_chr,last_chr,first_seen,last_seen,not_clreb

Rutine: memmove



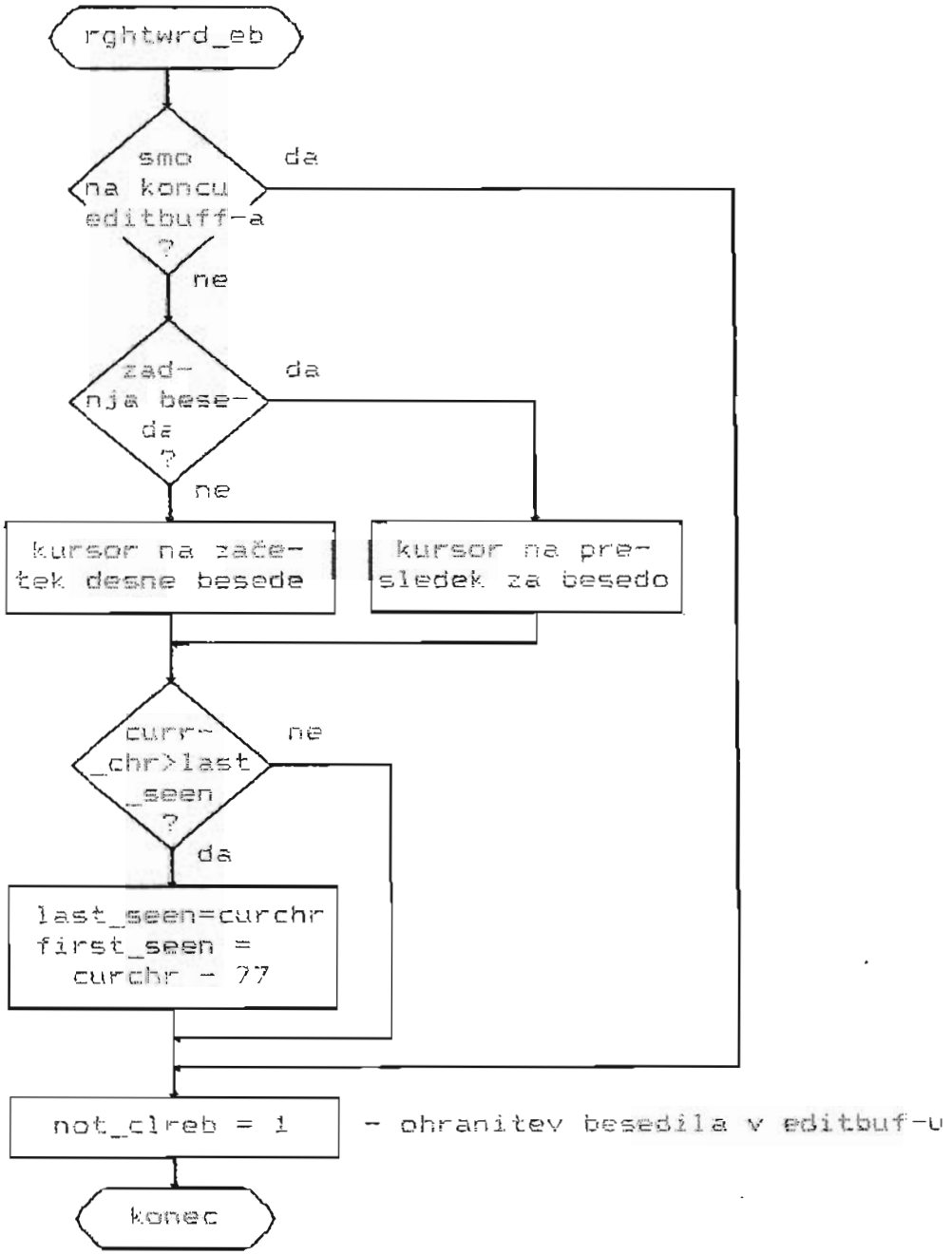
- ohranitev besedila v editbuf-u

rghtwrd_eb:

Pomik na začetek naslednje besede na desni strani. Če smo na zadnji besedi, se premaknemo na presledek za besedo. Če je nova pozicija izven prikazanega izreza editbuf-a,se edit polje scroll-ira v levo.

Sprememba: curr_chr,last_chr,first_seen,last_seen,not_clreb

Rutine: memmove

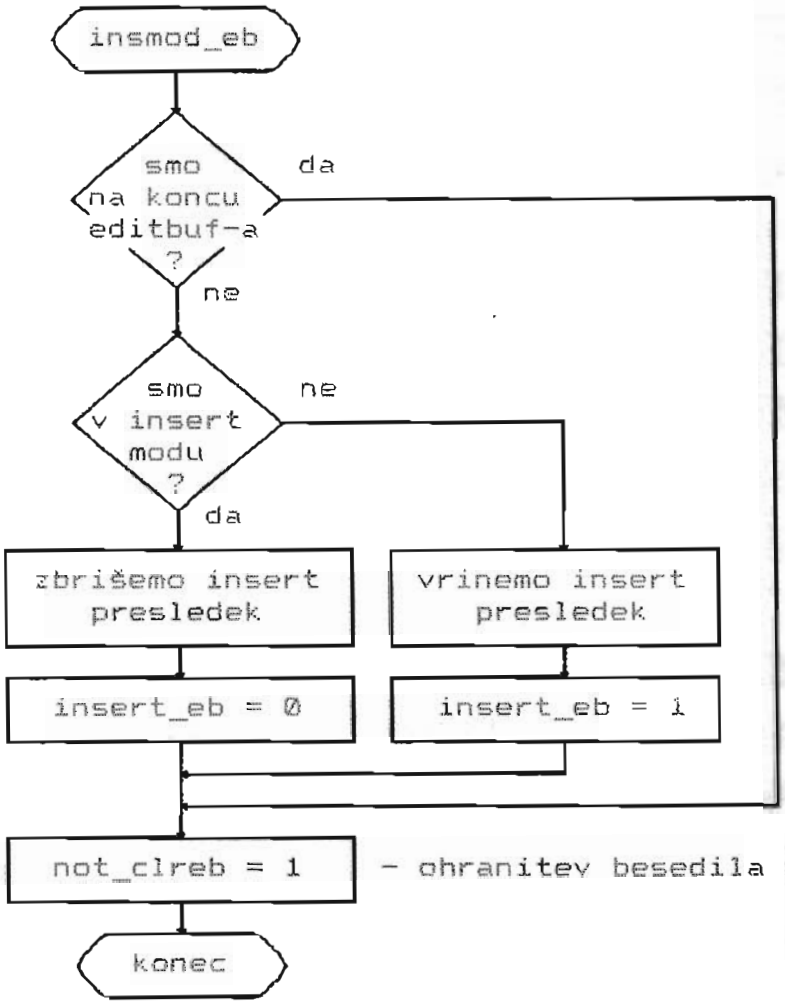


insmod_eb:

Preklopi iz insert v revise mode in obratno. Ob prihodu v insert način se na tekočo pozicijo vrine presledek (insert presledek), ki se v revise načinu spet zbriše. Če smo na koncu editbuf-a, ne moremo zamenjati načina. Podatek o načinu editiranja nosi globalna spremenljivka **insert_eb**.

Sprememba: last_chr, not_clreb

Rutine: memmove



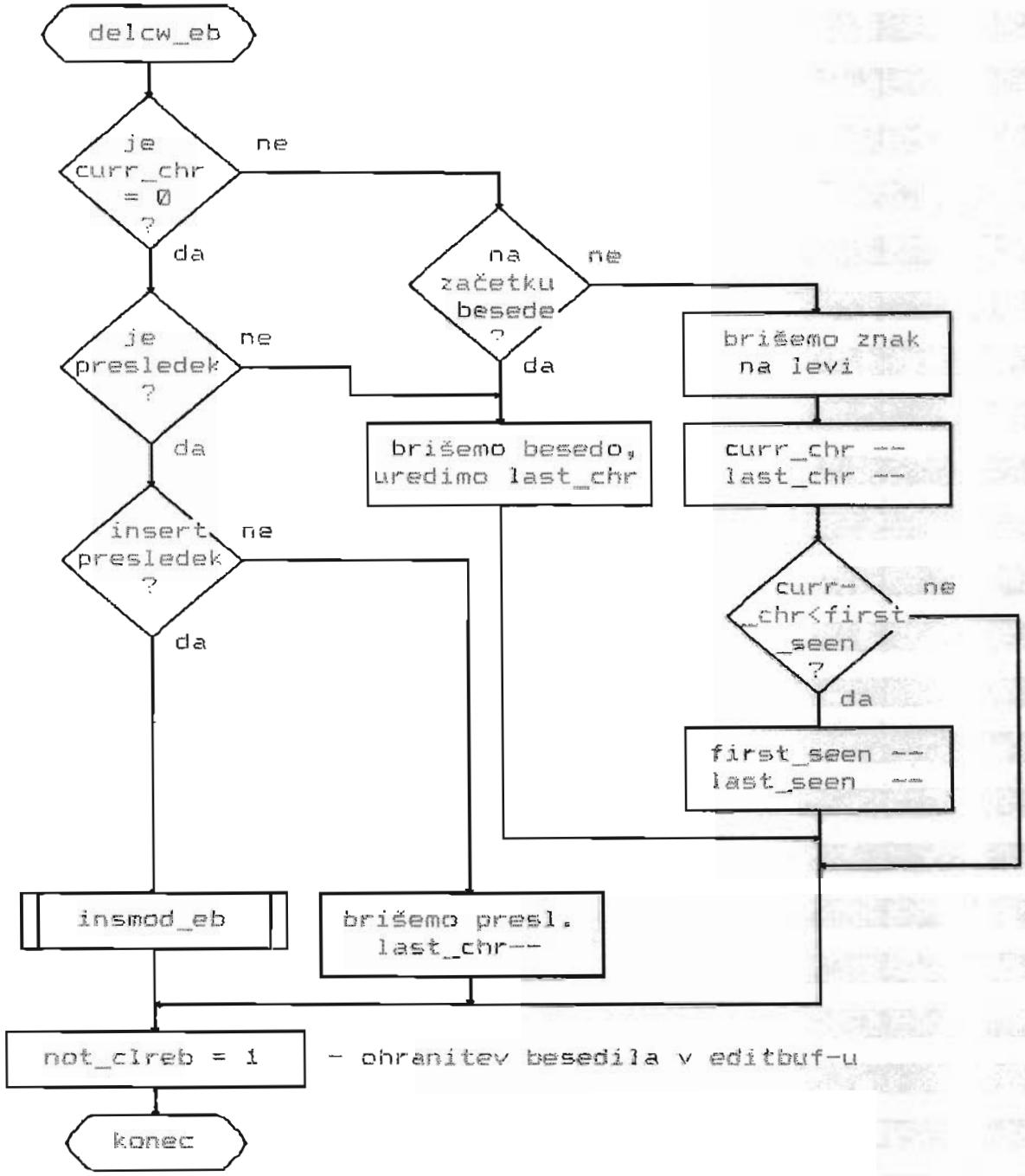
– ohranitev besedila v editbuf-u

delcw_eb:

Če je kursor na začetku besede, briše celo besedo. Če je kursor znotraj besede, briše znak pred kursorjem (kot backspace). Če je pri tem na začetku 16. vrste, ne pa tudi na začetku editbuf-a, scroll-ira edit polje v desno. Če je kursor na začetku edit polja in na presledku, ga briše. Če je bil to insert presledek, preklopi v revise način.

Sprememba: curr_chr,last_chr,first_seen,last_seen,not_clreb

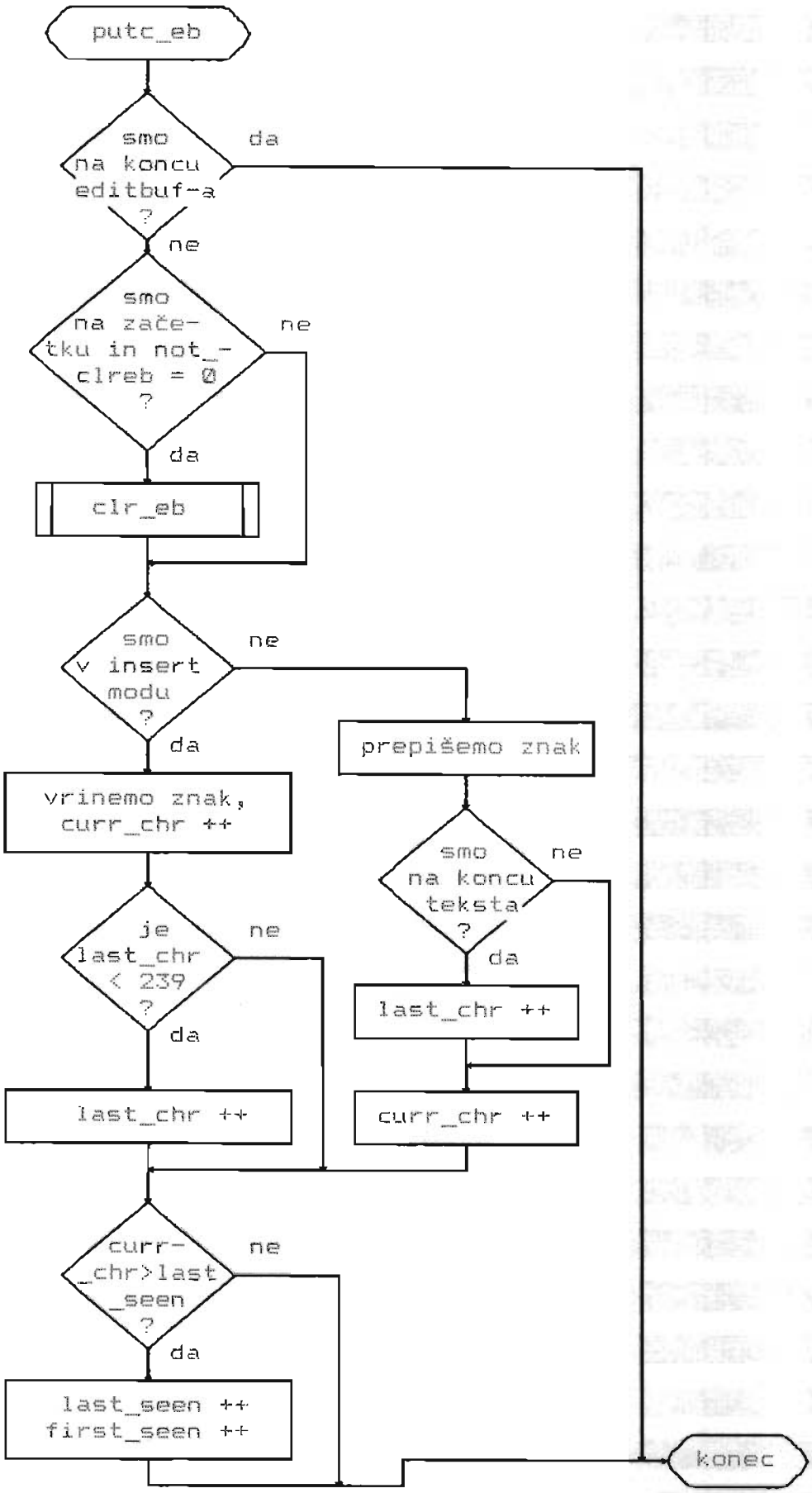
Rutine: memmove



putc_eb:

Vrine ali prepiše znak (odvisno od načina editiranja). Če je kursor na koncu editbuf-a, ni akcije. Če je kursor na koncu 17.vrstice, ne pa tudi na koncu editbuf-a, se prikaz premakne v desno (left scroll). Če je kursor na začetku edit polja in je v njem še staro besedilo, ki je ostalo po pritisku na CR, ga , če pred tem ni bila pritisnjena nobena editorska tipka (not_clred = 0), zbriše. V nasprotnem primeru staro besedilo ostane. Če vrivamo znake v polni editbuf, se izrinjeni znaki na koncu vmesnika izgubijo.

Vhod: znak za izpis
Sprememba: curr_chr,last_chr,first_seen,last_seen,not_clreb
Rutine: memmove

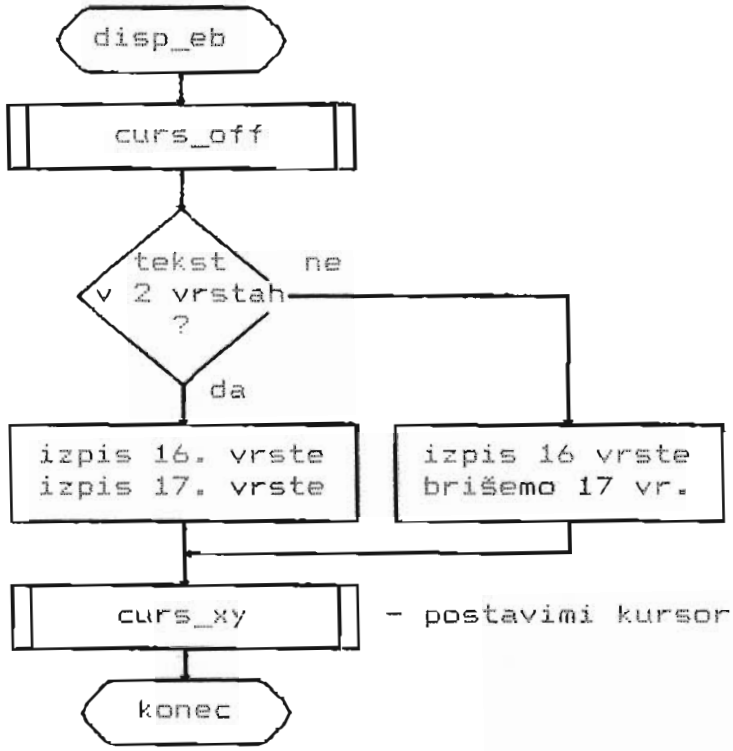

putc_eb
smo na koncu editbuf-a ?
da
ne
smo na začetku in not_clreb = 0 ?
ne
da
clr_eb
smo v insert modu ?
ne
da
prepišemo znak
vrinemo znak, curr_chr ++
smo na koncu teksta ?
ne
da
je last_chr < 239 ?
ne
da
last_chr ++
last_chr ++
curr_chr ++
curr_chr>last_seen ?
ne
da
last_seen ++
first_seen ++
konec

disp_eb:

Izpiše izrez 78 znakov editorskega vmesnika (od first_seen do last_seen) v edit polje ekrana (16. in 17 vrsta).

Mesto prvega znaka v 16. vrstici je rezervirano za znak <, ki pove, da se pred prvim znakom izpisanim v 16. vrstici nahaja besedilo. Mesto zadnjega znaka v 17. vrstici je rezervirano za znak >, ki pove, da se za zadnjim znakom izpisanim v 17. vrstici nahaja besedilo.

Rutine: putsn_xy, clreol, clr_line, curs_xy



- postavimi kursor na curr_chr

5. C R T - S I S T E M S K I K L I C I

Splošne zahteve za CRT program:

CRT driverski program naj ima naslednje lastnosti:

1). Vsebuje naj enotne rutine za brisanje, izpis, premike kursorja itd za celotno področje ekrana (ni več delitve na delovno in komunikacijsko polje ekrana).

2). Rutine morajo biti prirejene za klicanje iz C-ja in assemblerja. Za klice iz C-ja velja:

- vstopna točka rutine je ime rutine napisano z malimi črkami,
- vhodni parametri se prenašajo preko stack-a,
- izhodne vrednosti se vračajo v registru D7 ali preko podanih kazalcev.

Za klice iz asssemblerja velja :

- vstopna točka rutine je ime rutine napisano z velikimii črkami,
- vhodni parametri se prenašajo v registrih,
- izhodne vrednosti se vračajo v registrih.

3). Možno naj bo pisanje na video stran, ki se trenutno ne prikazuje (število obstoječih video strani je 8).

4). Pozicija kursorja in tekoča pozicija vpisovanja besedila morata biti medsebojno neodvisni.

- Vsaka video stran naj ima svoj kursor (s svojo obliko) in svojo tekočo pozicijo pisanja (s svojim atributom).
- Atribut besedila naj vsebuje podatke o velikosti znakov, jakosti znakov in o tem ali naj besedilo utripa.
- Nekemu že izpisanemu besedilu na ekranu naj bo mogoče zamenjati atribut.

KRATEK OPIS RUTIN:

Rutine za izbiro in aktivacijo strani:

get_dispg:

Vrne številko trenutno izbrane video strani za prikazovanje.
Izhod: številka trenutno prikazovane strani (word).

set_dispg:

Aktivira prikazovanje page strani na ekranu.
Vhod: page, številka strani, ki naj se prikazuje (word).

get_wpg:

Vrne številko trenutno izbrane video strani za vpisovanje.
Izhod: številka trenutno izbrane strani za vpisovanje (word).

set_wpg:

Aktivira page video stran za vpisovanje.
Vhod: page, številka strani, na katero želimo pisati (word).

Rutine za nastavitev atributa in tekoče pozicije:

get_att:

Vrne vrednost trenutnega atributa besedila na strani izbrani za vpisovanje.

Izhod: vrednost trenutnega atributa besedila (byte).

set_att:

Spremeni atribut besedila na strani izbrani za vpisovanje na vrednost att.

Vhod: att, vrednost željenega atributa vpisovanega besedila (byte).

chatrn:

Spremeni atribut n znakom napisanim za tekočo vpisovalno pozicijo, na strani izbrani za vpisovanje, na vrednost att.

Vhod: att, vrednost željenega atributa že vpisanega besedila (byte),
n, število znakov katerim naj se spremeni atribut(word),

chatrn_xy:

Spremeni atribut n znakom v vrsti y od kolone x naprej na strani izbrani za vpisovanje, na vrednost att.

Vhod: att, vrednost željenega atributa že vpisanega besedila (byte),
x, kolona začetka spremembe atributa (word),
y, vrsta začetka spremembe atributa (word),
n, število znakov katerim naj se spremeni atribut(word),

where_xy:

Vrne tekočo pozicijo pisanja na video strani izbrani za pisanje besedila.

Izhod: x, kolona trenutne pozicije za pisanje (word),
y, vrsta trenutne pozicije za pisanje (word).

goto_xy:

Postavi željeno tekočo pozicijo pisanja na video strani izbrani za pisanje besedila.

Vhod: x, kolona željene pozicije za pisanje (word),
y, vrsta željene pozicije za pisanje (word).

Rutine za izpis besedila:

putc_xy:

Izpiše ch znak z atributom att na x, y pozicijo strani izbrane za pisanje. Tekoča pisalna pozicija se premakne za izpisani znak oz. na začetek nove vrste, če je znak na koncu vrste.

Whod: att, atribut izpisa (byte),
x, kolona izpisa (word),
y, vrsta izpisa (word),
ch, znak za izpis (byte).

putc:

Izpiše ch znak s trenutno aktivnim atributom na tekočo pozicijo strani izbrane za pisanje. Tekoča pisalna pozicija se premakne za izpisani znak oz. na začetek nove vrste, če je znak na koncu vrste.

Vhod: ch, znak za izpis (byte).

puts_xy:

Izpiše string z atributom att na x, y pozicijo strani izbrane za pisanje. Tekoča pisalna pozicija se premakne za izpisani string oz. na začetek nove vrste. Znaki stringa, ki bi prišli čez konec vrste (40 znakov osnovne velikosti), se izgubijo.

Whod: att, atribut izpisa (byte),
x, kolona izpisa (word),
y, vrsta izpisa (word),
str, kazalec na string za izpis (long).

putsn_xy:

Izpiše nchr znakov stringa str z atributom att na x, y pozicijo strani izbrane za pisanje. Tekoča pisalna pozicija se premakne za izpisani string oz. na začetek nove vrste. Znaki stringa, ki bi prišli čez konec vrste (40 znakov osnovne velikosti), se izgubijo.

Whod: att, atribut izpisa (byte),
x, kolona izpisa (word),
y, vrsta izpisa (word),
str, kazalec na string za izpis (long),
nchr, število znakov, ki naj se izpišejo (word).

puts:

Izpiše string s trenutno aktivnim atributom na tekočo pozicijo strani izbrane za pisanje. Tekoča pisalna pozicija se premakne za izpisani string oz. na začetek nove vrste. Znaki stringa, ki bi prišli čez konec vrste (40 znakov osnovne velikosti), se izgubijo.

Whod: str, kazalec na string za izpis (long).

putsn:

Izpiše nchr znakov stringa str s trenutnim atributom na tekočo pozicijo strani izbrane za pisanje. Tekoča pisalna pozicija se premakne za izpisani string oz. na začetek nove vrste. Znaki stringa, ki bi prišli čez konec vrste (40 znakov osnovne velikosti), se izgubijo.

Whod: str, kazalec na string za izpis (long),
nchr, število znakov, ki naj se izpišejo (word).

Rutine za brisanje:

clrwpg:

Briše page video stran.

Vhod: page, številka video strani, ki naj se briše (word).

clrscr:

Briše ekran (video stran, ki se trenutno prikazuje).

clrline:

Briše vrstico line na strani izbrani za pisanje.

Vhod: line, številka vrstice, ki naj se briše (word).

clreol_xy:

Briše vrstico y od pozicije x do konca vrste na strani izbrani za pisanje.

Vhod: x, kolona (word),
y, vrsta (word).

clreol:

Briše od trenutno aktivne pozicije vpisovanja do konca vrstice na strani izbrani za pisanje.

Rutine kursorja:

curs_shape:

Spremeni obliko in jakost kursorja glede na vrednost shape na strani izbrani za pisanje.

Vhod: shape, oblika in jakost kursorja (word).

curs_xy:

Postavi kursor na pozicijo x, y in ga prižge.

Vhod: x, kolona nove pozicije kursorja (word),
y, vrsta nove pozicije (word).

curs_on:

Prižge kursor.

curs_off:

Ugasne kursor.

get_curs:

Vrne trenutno pozicijo kursorja na strani, ki se prikazuje.

Izhod: x, kolona trenutne pozicije kursorja (word),
y, vrsta trenutne pozicije kursorja (word).

6. S I S T E M S K I P R I K A Z I

Sistemski prikazi so izpisi, ki se pojavljajo na ekranu sistema LJUMO. Delimo jih lahko na osnovne in neposredne sistemske prikaze.

OSNOVNI SISTEMSKI PRIKAZI

So prikazi, ki se izpišejo ob prihodu v nek režim ali podrežim sistema. Nahajajo se vedno na prvi video strani. Vse osnovne prikaze sestavljajo enaka osnovna področja ekrana :

- vrsta za izpis glavnega režima in podrežima (1.vrsta),
- diagnostična vrstica, za prikaz najpomembnejšega sporočila (2. vrsta),
- vrsta za izpis nižjih podrežimov in aktivnega programa (3.vrsta),
- delovno polje, proste vrste za prikaz (4.- 14. vrsta),
- help vrstica, za pomožno besedilo pri dialogu (15.vrsta),
- editorsko polje (16. in 17. vrsta),
- polje softkey tipk (18. - 20. vrsta).

Za pomoč pri pisanju osnovnih prikazov obstaja nekaj osnovnih pomožnih rutin.

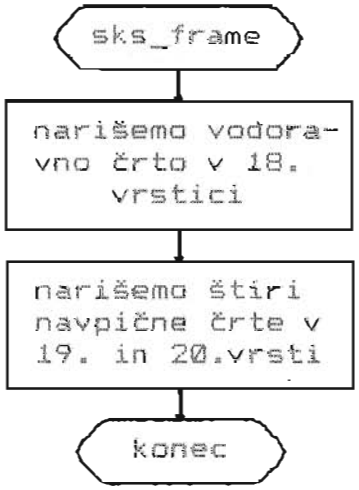
Pomožne rutine osnovnih sistemskih prikazov:

Nekaj rutin je bilo opisanih že v prejšnjih poglavjih, npr.:

- disp_al, prikaz najpomembnejšega alarma v diagnostični vrstici,
- clr_eb, brisanje editorskega polja in vmesnika.

sks_frame:

Izpiše okvir softkey tipk.
Rutine: goto_xy, putc, putc_xy



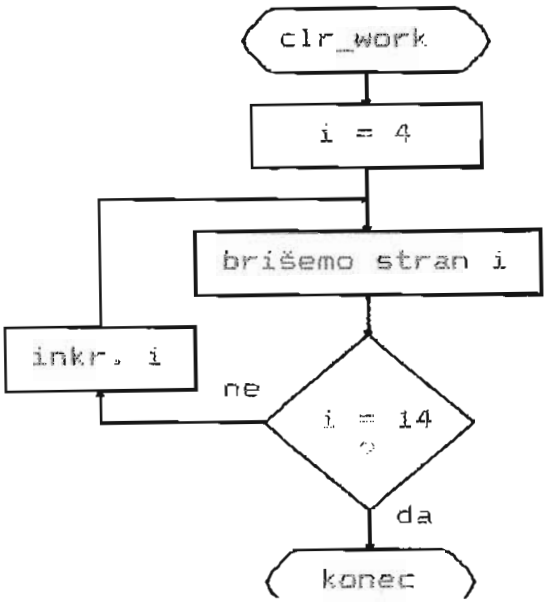
sks_disp:

Izpiše curr_sks stran softkey stringov tekočega stanja v softkey polja.

clr_work:

Briše delovno polje ekrana (4.- 14.vrste) video strani izbrane za pisanje.

Rutine: clrline



disp_cstate:

Izpiše ime trenutnega stanja (režima ali podrežima) na mesto ekrana, ki mu glede na globino stanja pripada. Razpored napisov na ekranu oz. video strani:

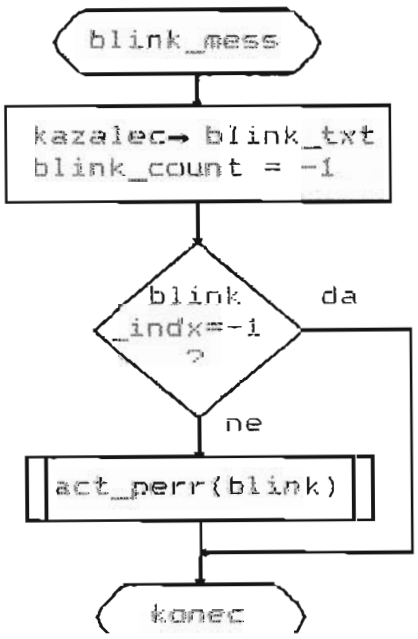
- glavni režim (vrsta = 1, kolona = 1),
- 1. podrežim (vrsta = 1, kolona = 25),
- 2. podrežim (vrsta = 3, kolona = 1),
- 3. podrežim (vrsta = 3, kolona = 25),
- 4. in nižji podrežimi se prepisujejo čez mesto 3.

Podatek o globini podrežima dobi iz cstate indeksa. Številko stanja prebere v vmesniku statebuf. ASCII string imena dobi iz tabele stingov **SSTATE_TAB** (številka stanja je tudi indeks stringa v tabeli).

blink_mess:

Naslov sporočila zapiše v kazalec **blink_txt**, inicializira števec **blink_cout** (word) na -1 (FFFF) in aktivira permanentno periodično rutino **blink**, če je **blink_indx** = -1 (še ni aktivna).

Vhod: naslov besedila, ki naj utripa,
Sprememba:blink_txt, blink_indx
Rutine: act_perr, blink



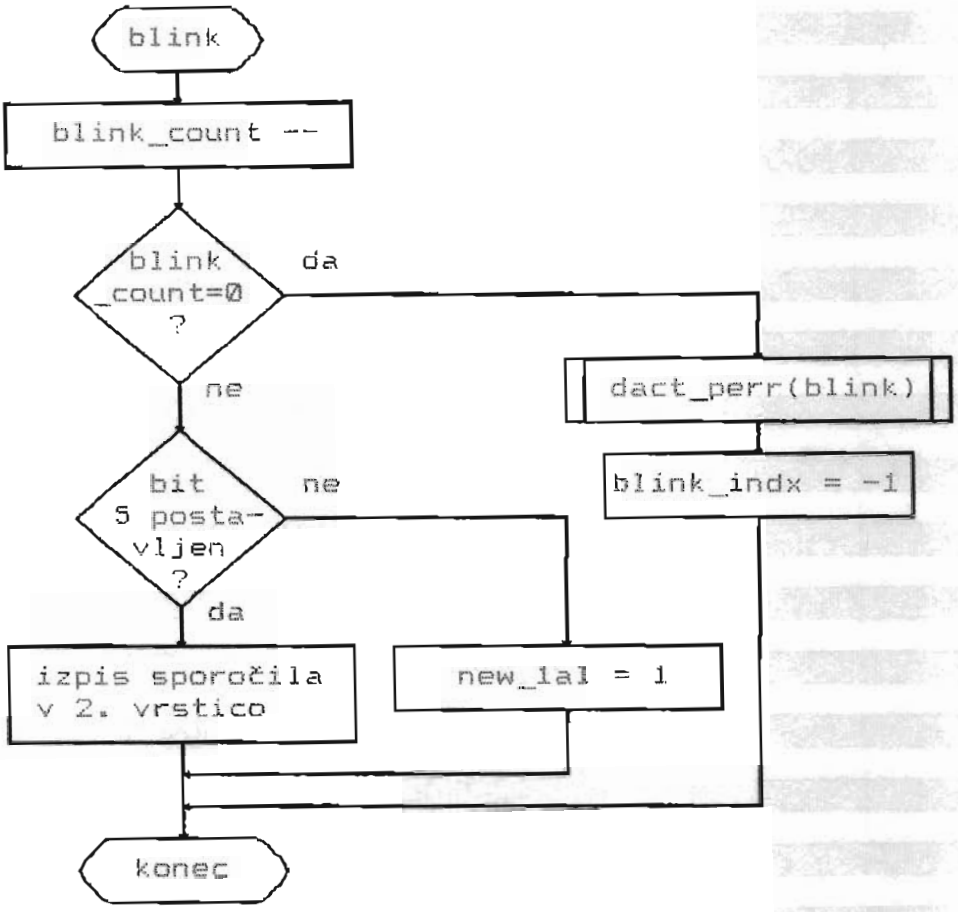
blink:

Je periodična rutina, ki dekrementira števec blink_count in testira njegov bit 5. Če je postavljen, izpiše sporočilo v diagnostično vrstico. Če je podrt, zapiše 1 v new_1al, kar povzroči izpis najpomembnejšega alarma v diagnostično vrstico. Napis prvega alarma sistema in utripajočega sporočila se izmenjujeta na približno:

30 * 2**5 ms = 0,96 s. .

Ko je blink_count enak 0, se rutina deaktivira (⟶ sporočilo utripne štirikrat). Po končanem utripanju ostane najpomembnejši alarm v diagnostični vrstici nespremenjen.

Vhod: blink_txt,
Sprememba:blink_count, blink_indx, new_1al
Rutine: puts_xy



NEPOSREDNI SISTEMSKI PRIKAZI:

Poleg osnovnih prikazov obstajajo v sistemu LJUMO tudi t.i. neposredni prikazi (npr. prikaz trenutnih pozicij, prikaz aktivnih NC funkcij, prikaz vhodno-izhodnih signalov sistema).

Vsak neposredni prikaz se nahajajo na svoji video strani. Izbran neposredni prikaz je dostopen v vseh režimih in podrežih dela, v vsakem trenutku, preko tipke PAGE. Pritisk na to tipko povzroči zamenjavo trenutnega osnovnega prikaza z izbranim neposrednim in obratno. Neposredni prikaz, ki naj se izmenjuje z osnovnim, izberemo s SHIFT PAGE tipko. Izbira novega neposrednega prikaza povzroči zapis številke njegove video strani v spremenljivko **bgnd_page**, aktiviranje periodične rutine tega prikaza in prikaz njegove video strani na ekranu. Periodična rutina osvežuje prikaz dinamičnih vrednosti na ekranu (npr. pozicij orodja).

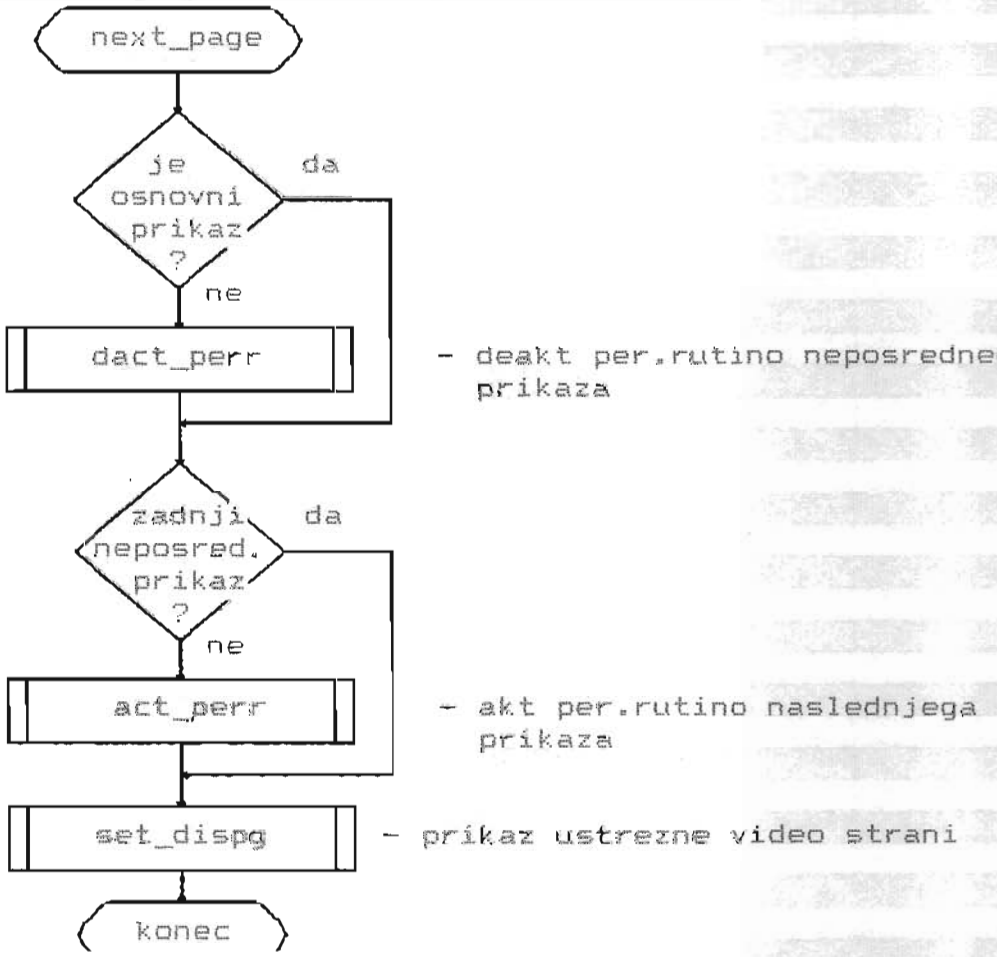
Kazalci na periodične rutine neposrednih prikazov se nahajajo v tabeli **bgruts_tab**. Spremenljivka bgnd_page vsebuje številko izbranega neposrednega prikaza, **bgnd_indx** pa vsebuje index periodične rutine trenutno aktivnega neposrednega prikaza.

next_page:

Ob pritisku na tipko SHIFT PAGE prikaže naslednji neposredni prikaz, oziroma osnovnega, če pride do konca.

Sprememba:bgnd_page, bgnd_indx

Rutine: dact_perr, act_perr, set_dispg

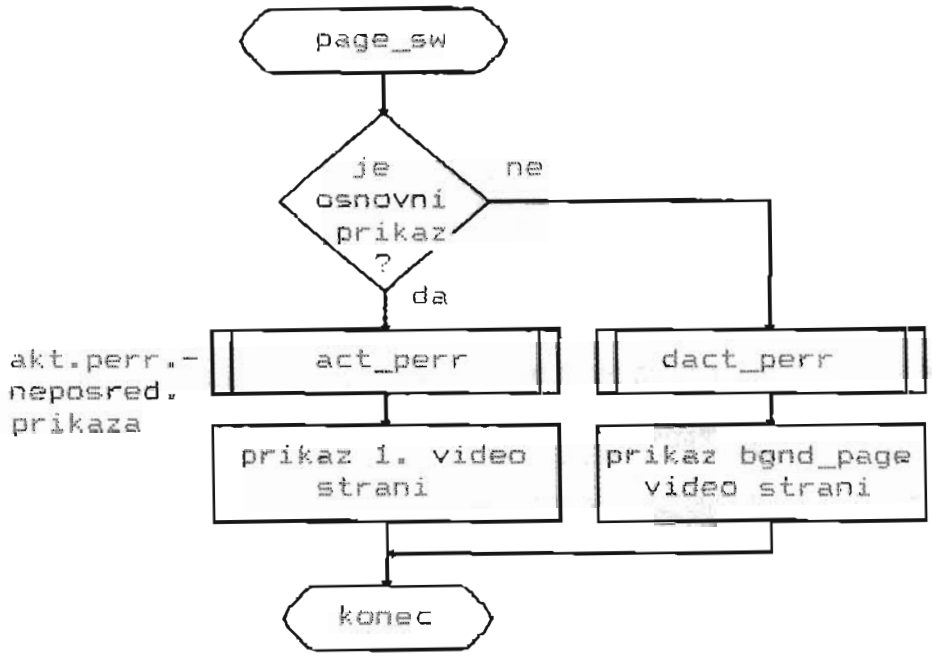


page_sw:

Izvede se ob pritisku PAGE tipke. Osnovni prikaz zamenja z izbranim neposrednim (bgnd_page) in obratno.

Sprememba: bgnd_indx

Rutine: dact_perr, act_perr, set_dispg



PRINCIP IZRAČUNA OW POZICIJ:

Osnovna pozicija na obdelovalnem stroju je trenutna pozicija referenčne točke revolverske glave glede na ničelno točko stroja (OM pozicija). Največkrat pa nas zanima trenutna pozicija konice orodja glede na ničelno točko obdelovanca (OW pozicija). OW pozicija se zato prikazuje v več osnovnih in neposrednih prikazih.

OW pozicijo v neki osi (npr X) dobimo preko enačbe:

$$OW(X) = OM(X) - ZOFF(X) - TCORR(X)$$

,kjer je:

- OW(X) pozicija konice orodja glede na ničelno točko obdelovanca v smeri osi X,
- OM(X) pozicija referenčne točke revolverske glave glede na ničelno točko stroja v smeri osi X,
- ZOFF(X) odmik izbrane referenčne točke obdelovanca od referenčne točke stroja v smeri osi X,
- TCORR(X) odmik konice orodja od referenčne točke revolverske glave v smeri osi X.

7. D I A L O G

Dialog sistema LJUMO omogoča uporabniku enostavnejše, in hitrejše delo. Glavni del dialoga predstavljajo opcije zapisane na softkey tipkah ali pa v delovnem polju ekrana (od 4. do 14. vrstice), ki jih uporabnik lahko izbira. Način izbiranja in rezultirajoče akcije so odvisne od vrste dialoga.

Poleg softkey dialoga, kjer izbiramo opcije s pritiskom na ustrezne softkey tipke, obstajata še dve vrsti dialoga:

- **choose** dialog,
- **next** dialog.

Pri choose dialogu izbiramo poljubne opcije v delovnem polju ekrana s pritiskom na choose softkey tipko. Pri next dialogu imamo lahko poleg poljubnih opcij tudi obvezne. Dokler tekoče obvezne opcije nismo izbrali, s tipko next ne moremo izbirati naslednjih.

CHOOSE DIALOG

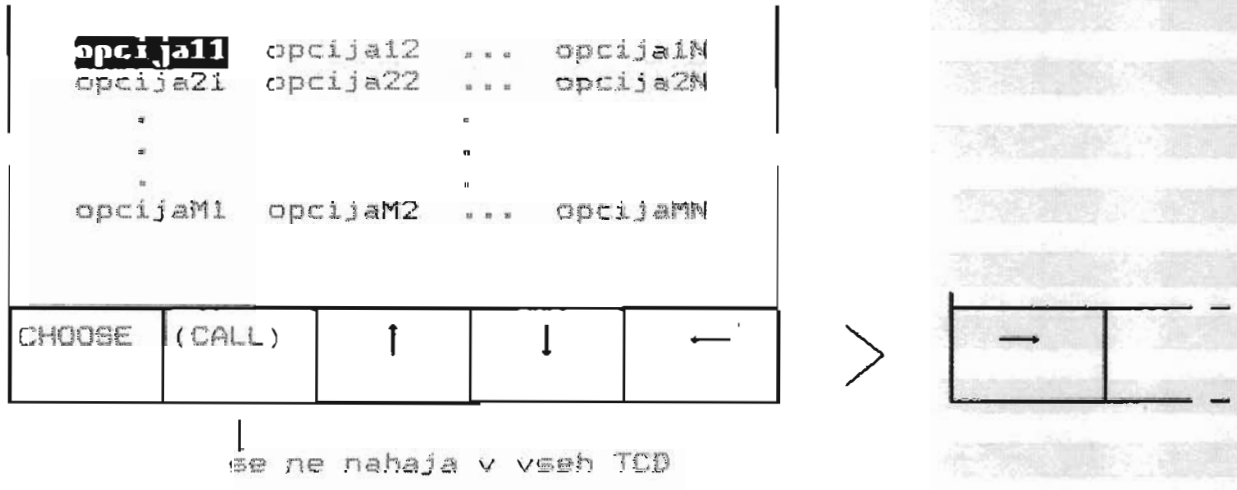
Choose dialog ima ime po tipki za izbiro opcije na delovnem področju ekrana. Med opcijami dialoga se gibljemo s pomočjo softkey tipk ↓, ↑, →, ←. Pritisk na tipko choose prenese izbrano opcijo na tekočo pozicijo v editorskem polju. Način aktivacije izbrane opcije je odvisen od tipa choose dialoga. Obstajata dva tipa choose dialoga:

- **table** tip,
- **parent** tip.

Glavna razlika med obema je v tem, da table tip vsebuje opcije, ki jih naloži na ekran iz konstantne tabele, parent tip pa vsebuje dinamične opcije (npr. imena obstoječih NC podprogramov), ki jih poišče v sistemu. Razlike so tudi v akcijah. Parent tip odpre podrežim z novim dialogom (opravlja funkcijo prehajanja stanj), table tip pa le opravi neko operacijo nad izbranim podatkom.

Table choose dialog (TCD)

Splošni primer oblike ekrana pri TCD:



S puščicami osvetlimo željeno opcijo. S tipko CHOOSE jo prepišemo na mesto kursorja v edit polju. Željeno opcijo lahko žapišemo tudi direktno preko alfanumeričnih tipk posluževalnega panoja. Opcijo aktiviramo s tipko CALL, ENTER ali pa aktivacija ni možna (odvisno od podrežima).

Splošna oblika tabele opcij je :

```
DC.B    'opcija11',0,'opcija12',0,...,'opcija1N',0,0Ah,
DC.B    'opcija21',0,'opcija22',0,...,'opcija2N',0,0Ah,
          .                             .
          .                             .
          .                             .
DC.B    'opcijaM1',0,'opcijaM2',0,...,'opcijaMN',0,0Ah,
DC.B    0
```

Konec stringa opcije označuje 0. Konec vrstice označuje znak Newline (0Ah). Konec tabele označuje prazni string t.j 0 na začetku nove vrste.

Splošne TCD rutine:

Za izpis tabele na ekran, gibanje po tabeli in izbiro opcije pri TCD obstajajo naslednje splošne rutine:

tchoose_ini:

- prepiše opcije iz tabele katere naslov je v **tchoose_tab** v delovno polje ekrana,
- inicializira kazalca aktivne opcije na ekranu (**tchoose_x**, **tchoose_y**) ter kazalec aktivne opcije v tabeli (**tctab_curr**) in osvetli prvo opcijo na ekranu .

tchoose_up:

Aktivira se ob pritisku ↑ softkey tipke. Osvetli prvo opcijo predhodne vrstice, uredi kazalca aktivne opcije na ekranu (**tchoose_x**, **tchoose_y**) ter kazalec aktivne opcije v tabeli (**tctab_curr**). Če smo bili ob pritisku ↑ v prvi vrsti opcij, ni akcije. Če smo bili ob pritisku ↑ v prvi vrsti delovnega polja ekrana (4. vrstica), ne pa tudi v prvi vrsti opcij, se prikaz premakne za eno vrsto višje (down scroll).

tchoose_dwn:

Aktivira se ob pritisku ↓ softkey tipke. Osvetli prvo opcijo naslednje vrstice, uredi kazalce tchoose_x, tchoose_y in tctab_curr. Če smo bili ob pritisku ↓ v zadnji vrsti opcij ni akcije. Če smo bili ob pritisku ↓ v zadnji vrsti delovnega polja ekrana (14. vrsta), ne pa tudi v zadnji vrsti opcij, se prikaz premakne za eno vrsto navzdol (up scroll).

tchoose_pgup:

Je akcijska rutina, ki se aktivira ob pritisku /D posluževalne tipke. Prikaže predhodno stran opcij. Če je ostalo premalo predhodnih vrstic za celo novo stran, prikaže ekran od prve vrstice opcij naprej. Po izpisu nove strani je osvetljena prva opcija v 9. vrsti ekrana (t.j. v srednji vrstici delovnega polja ekrana). Če smo bili ob pritisku tipke v prvi vrsti opcij, ni akcije.

tchoose_pgdwn:

Je akcijska rutina, ki se aktivira ob pritisku Đ/ posluževalne tipke. Prikaže naslednjo stran opcij. Če je ostalo premalo predhodnih vrstic za celo novo stran, prikaže ekran do zadnje vrstice opcij. Po izpisu nove strani je osvetljena prva opcija v 9. vrsti ekrana. Če smo bili ob pritisku te tipke v zadnji vrsti opcij, ni akcije.

tchoose_left:

Aktivira se ob pritisku ← softkey tipke. Osvetli levo opcijo na ekranu, uredi kazalce tchoose_x, tchoose_y in tctab_curr. Če smo na prvi opciji tabele, ni akcije. Če smo na prvi opciji neprve vrstice, osvetli zadnjo opcijo predhodne vrstice. Če smo pri tem še v prvi vrsti delovnega polja, scroll-ira ekran.

tchoose_right:

Aktivira se ob pritisku → softkey tipke. Osvetli desno opcijo na ekranu, uredi kazalce tchoose_x, tchoose_y in tctab_curr. Če smo na zadnji opciji tabele, ni akcije. Če smo na zadnji opciji nezadnje vrstice, osvetli prvo opcijo naslednje vrstice. Če smo pri tem še v zadnji vrsti delovnega polja, scroll-ira ekran.

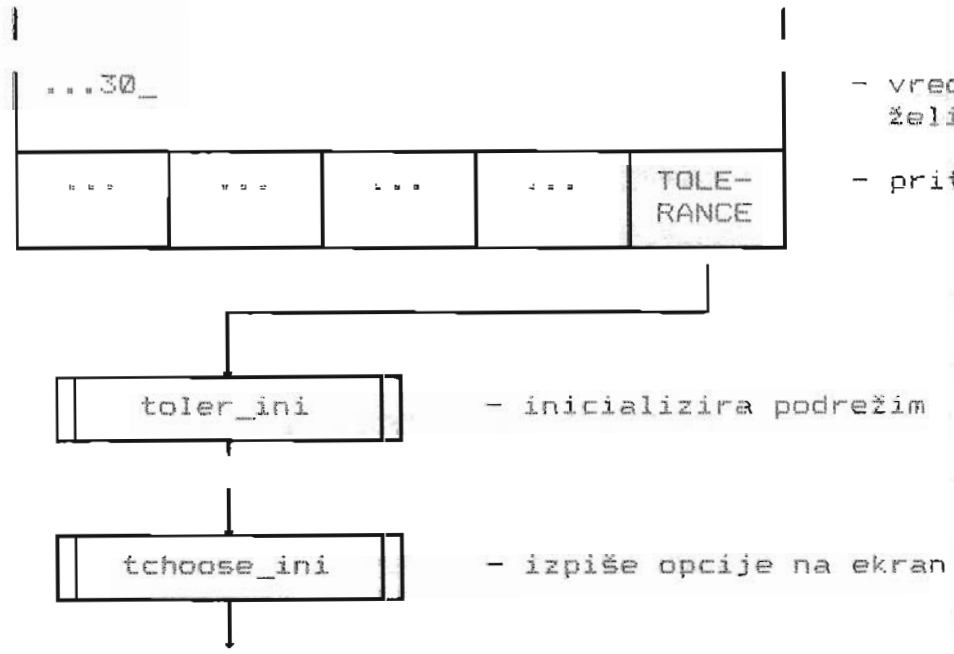
tchoose:

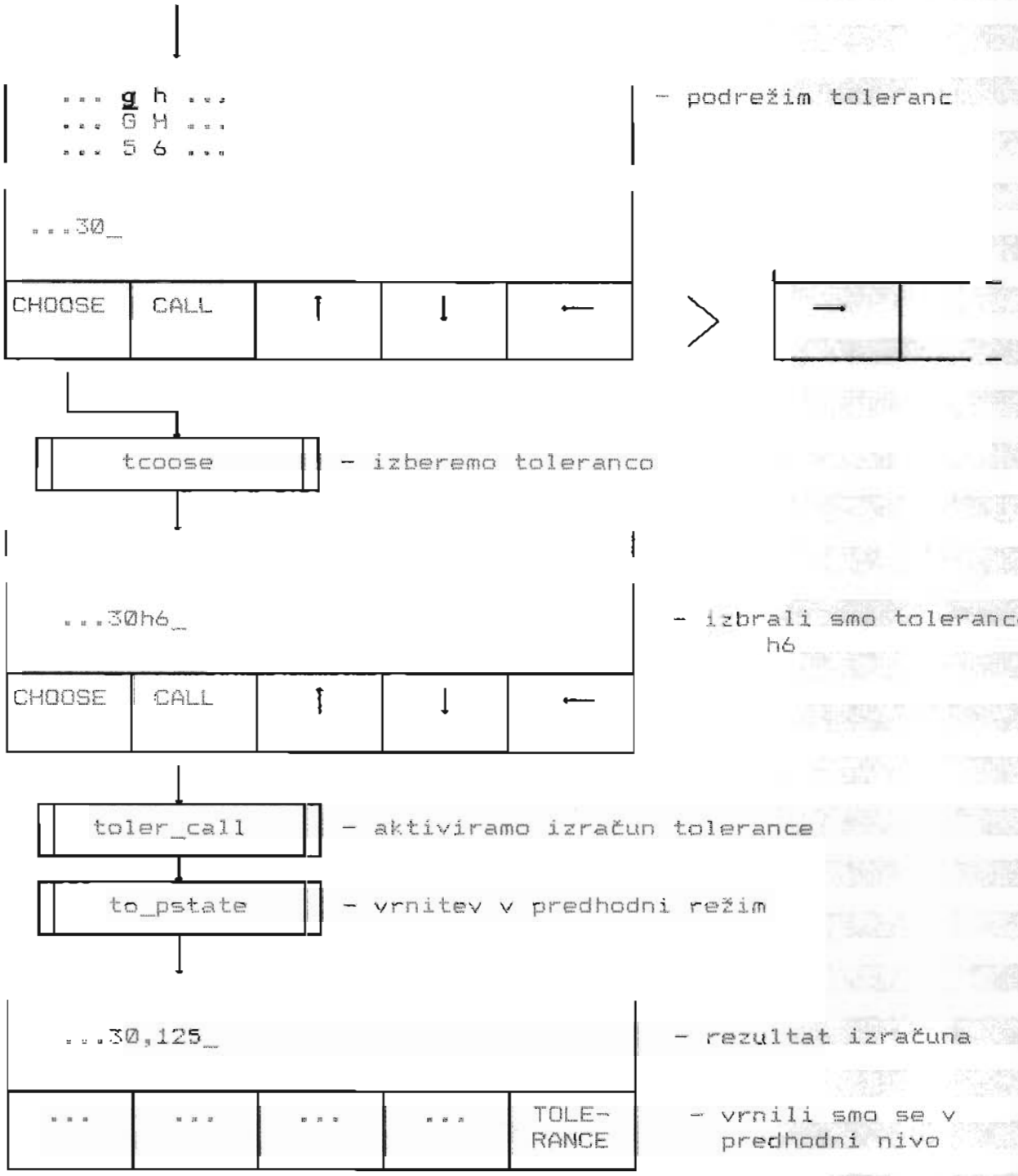
Aktivira se ob pritisku CHOOSE tipke. Prepiše osvetljeno opcijo na mesto kursorja v edit polju. Če v editbuf-u ni dovolj prostora ni akcije. Če smo na koncu edit polja v editbuf-u pa je še dovolj prostora za izpis opcije, scroll-ira edit polje.

Rutine: puts_eb

Primer izbire tolerance:

TCD je predviden v podrežimu toleranc, kalkulatorja, aritmetike in pri izbiri G funkcij. Oglejmo si primer izbire tolerance:





toler_ini:

- zapiše podatke stanja na state_stack,
- inicializira delovno polje ekrana,
- izpiše ime podrežima na ekran,
- naslov tabele dialoga toleranc **TOLERANCE_TCD** zapiše v kazalec tabele TCD dialoga **tchoose_tab**,
- pokliče rutino **tchoose_ini**, ki izpiše opcije iz tabele tchoose_tab na ekran.

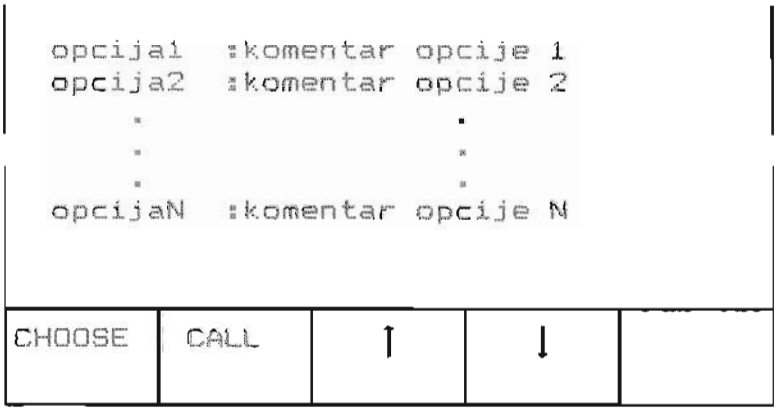
toler_call:

- izračuna toleranco zapisano v besedi pred kursorjem,
- če je izračun uspel, aktivira rutino za vrnitev v predhodni režim (to_pstate).

Postopek izbire pri kalkulatorju, aritmetiki, G funkcijah itd je enak. Razlika je le v tem, da pri kalkulatorju aktiviramo izračun izbranega matematičnega izraza s pritiskom na tipko ENTER, pri aritmetiki in G funkcijah pa ni aktivacije (opcije samo izbiramo oz. vpisujemo v program).

Parent choose dialog (PCD)

PCD se uporablja v podrežimu CYCLE in SUBPROG pri editiranju NC progama. Opcije (imena ciklov oz. drugih podprogramov), ki se nahajajo v RAMu sistema, se zapišejo na ekran v naslednji obliki:



Komentar opcije je v v bistvu komentar prepisan iz prvega bloka podprograma (če le-ta obstaja).

S puščicama osvetlimo željeno opcijo. S tipko CHOOSE jo prepišemo na mesto kursorja v edit polju. Opcija se aktivira s tipko CALL. Željeno opcijo je možno tudi direktno zapisati s pomočjo alfanumeričnih tip posluževalnega panoja in aktivirati s tipko CALL.

Aktivacija je možna le, če izbrani podprogram vsebuje bloke NEXT dialoga (bloki N9000 do N9050). Aktivacija odpre podrežim z NEXT dialogom v obliki, ki je zapisana v NEXT blokih.

Rutine za izpis opcij na ekran, gibanje med opcijami in izbiro opcije (**pchoose_ini**, **pchoose_up**, **pchoose_dwn**, **choose_pgup**, **pchoose_pgdw** in **pchoose**) delujejo podobno kot pri TCD, le da morajo opcije iskati po uporabniškem RAM-u.

Primer zbire podprograma

- pritisnemo tipko

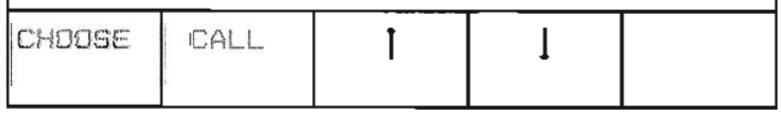
subprog_ini - inicializira podrežim

subprog_list - poišče in izpiše podprograme na ekran

L001 : PODPROGRAM ZA ...
L060 : PODPROGRAM KI ...

pchoose - ime izbranega podpr. prepiše na kursor pozicijo edit polja

...L100_



subprog_call - kazalec na začetek izbranega podprog. zapiše v **nextd_prog** in pokliče rutino find_nextd

find_nextd - poišče NEXT dialog bloke v izbranem podprogramu. Če obstajajo, odpre nov podrežim z NEXT dialogom.

- nov podrežim z next dialogom

NEXT DIALOG

NEXT dialog (ND) ima ime po tipki za prehod na naslednjo opcijo (NEXT). ND se aktivira v podrežimih s Parent choose dialogom, pri izbiranju določenih programov oz. podprogramov.

Glavna lastnost ND je v tem, da so vsi podatki za dialog shranjeni v t.i. next blokih (N9000 - N9050) izbranega programa. Če program ne vsebuje next blokov, se ND ne odpre.

Pri editiranju programov z next bloki so le-ti vidni le, če je v Setting data podatkih postavljen določen flag. Tako so podatki o dialogu zavarovani pred slučajnimi spremembami.

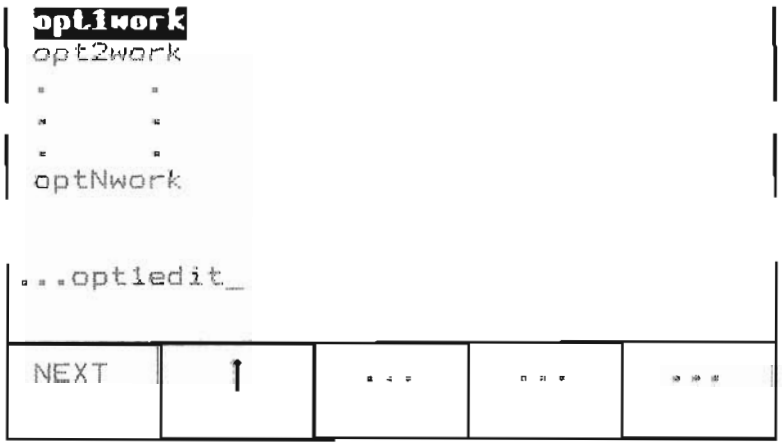
Oblika next bloka v NC programu je naslednja:

N9000 opttype, optedit, optwork, optprog

, kjer je:

- opttype, znak obveznost opcije (1-obvezna, 0-neobv.),
- optedit, besedilo opcije, ki se prepiše v edit vrsto,
- optwork, besedilo opcije, ki se vpiše v delovno polje ekrana,
- optprog, besedilo opcije, ki se vpiše v NC program ob pritisku ENTER tipke.

Primer oblike ekrana pri ND



Ustrezni string za izpis v edit polje (optXedit), ki pripada osvetljeni opciji v delovnem polju (optXwork), se avtomatično izpiše na pozicijo kursorja v edit polju.

Če je osvetljena opcija obvezna, z NEXT tipko ne moremo izbrati naslednje, preden ji v edit polju nismo dopisali numerične vrednosti (s tem je opcija obdelana). Če osvetljena opcija ni obvezna, lahko gremo s tipko NEXT na naslednjo. Ko osvetlimo naslednjo opcijo v delovnem polju, se čez staro optXedit besedilo v edit polju izpiše optX+1edit besedilo.

Če želimo obdelati neobvezno opcijo, ki smo jo preskočili, si pomagamo s softkey tipko ↑. Pritisk na to tipko osvetli opcijo v predhodni vrstici.

Ko obdelamo vse obvezne opcije, s pritiskom na tipko ENTER sprožimo prepis vrednosti iz edit polja v NC program, ki ga editiramo. Pri prepisu se stringi optXedit zamenjajo s stringi za vpis v NC program (optXprog).

Splošne rutine next dialoga

Za iskanje next blokov v programu, izpis opcij ND na ekran in gibanje med opcijami obstajajo naslednje splošne rutine:

find_nextd:

Poišče NEXT bloke v programu katerega naslov je v **next_prog**. Če obstajajo, zapiše naslov prvega next bloka v spremenljivko **nexd_curr** in pokliče rutino **nextd_ini**.

nextd_ini:

- inicializira podrežim z next dialogom (shrani podatke stanja na state_stack, izpiše podrežim na ekran in izpiše softkey stringe),
- inicializira kazalec vrstice aktivne opcije na ekranu (**nextd_y**),
- izpiše prvo stran opcij na ekran,
- osvetli prvo opcijo v delovnem polju ekrana,
- prepiše optXedit string aktivne opcije na mesto kursorja v edit polju.

Nextd:

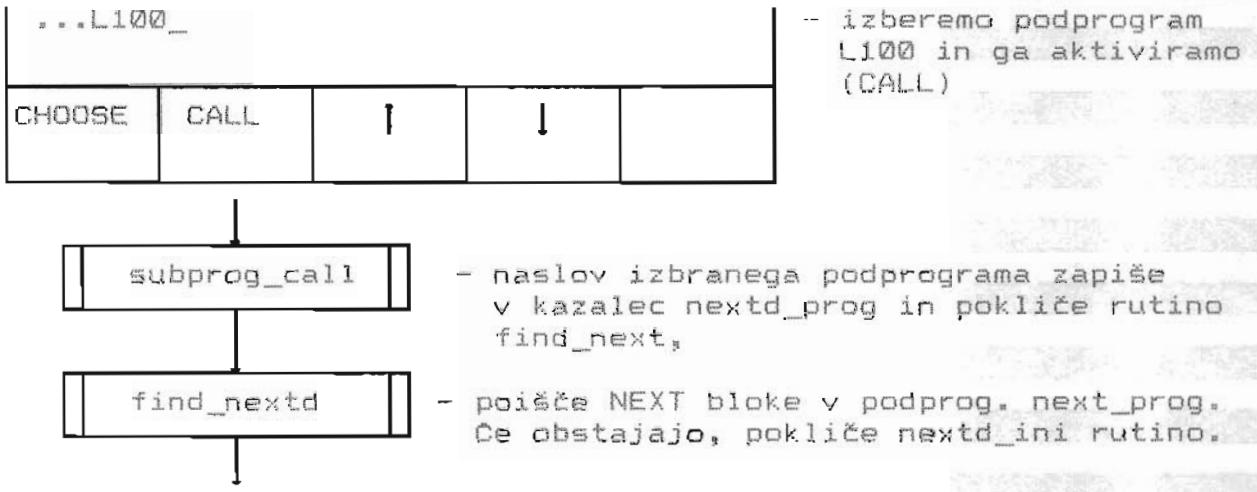
Aktivira se ob pritisku NEXT softkey tipke. Če je osvetljena opcija obvezna in ji v edit polju še nismo dopisali numerične vrednosti, ni akcije. V nasprotnem primeru osvetli opcijo v naslednji vrstici delovnega polja, v editorsko polje vpiše edit string in uredi kazalca nextd_curr in nextd_y. Če smo bili v zadnji vrstici delovnega polja, ne pa tudi na zadnji opciji dialoga, scroll-ira ekran.

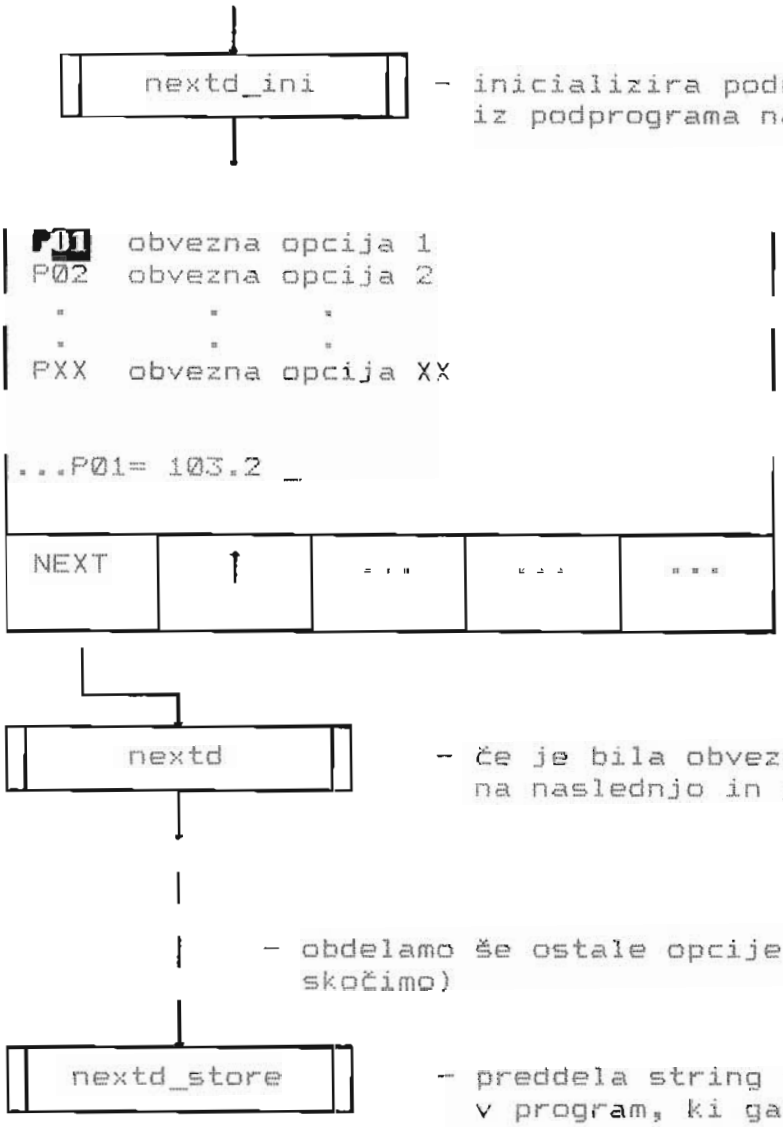
nextd_up:

Aktivira se ob pritisku ↑ softkey tipke. Osvetli prvo opcijo predhodne vrstice, v editorsko polje vpiše njen edit string, uredi kazalec vrstice aktivne opcije na ekranu (nextd_y) in kazalec NC bloka z aktivno opcijo (nextd_curr).

Če smo bili ob pritisku ↑ na prvi opciji dialoga, ni akcije. Če smo bili ob pritisku ↑ v prvi vrsti delovnega polja ekrana (4. vrstica), ne pa tudi na prvi opciji dialoga, se prikaz premakne za eno vrsto višje (down scroll).

Primer izbire v next dialogu





nextd_store:

Aktivira se ob pritisku na ENTER tipko. Če smo obdelali vse obvezne opcije in je zapis v edit vrsti brez sintaktičnih napak, zamenja optXedit stringe z optXprog stringi ter besedilo zapiše v editirani NC program.

Next dialog deluje zelo podobno v vseh podrežimih, kjer je uporabljen.

8. OBLIKA IN EDITIRANJE NC PROGRAMOV

V sistemu LJUMO obstaja več vrst programov, ki jih lahko delimo v dve glavni skupini:

- uporabniški programi,
- sistemski programi.

Med uporabniške programe lahko štejemo:

- glavne NC programe,
- NC podprograme,
- programe zero offset-ov,
- programe podatkov orodij.

Med sistemske programe lahko štejemo:

- programe strojnih parametrov,
- programe PC sporočil.

Uporabniški programi se nahajajo v uporabniškem področju RAM-a. Vsi uporabniški programi imajo v RAM-u enako fizično obliko. Sestavljeni so iz blokov, ki so označeni s črko N in številko bloka.

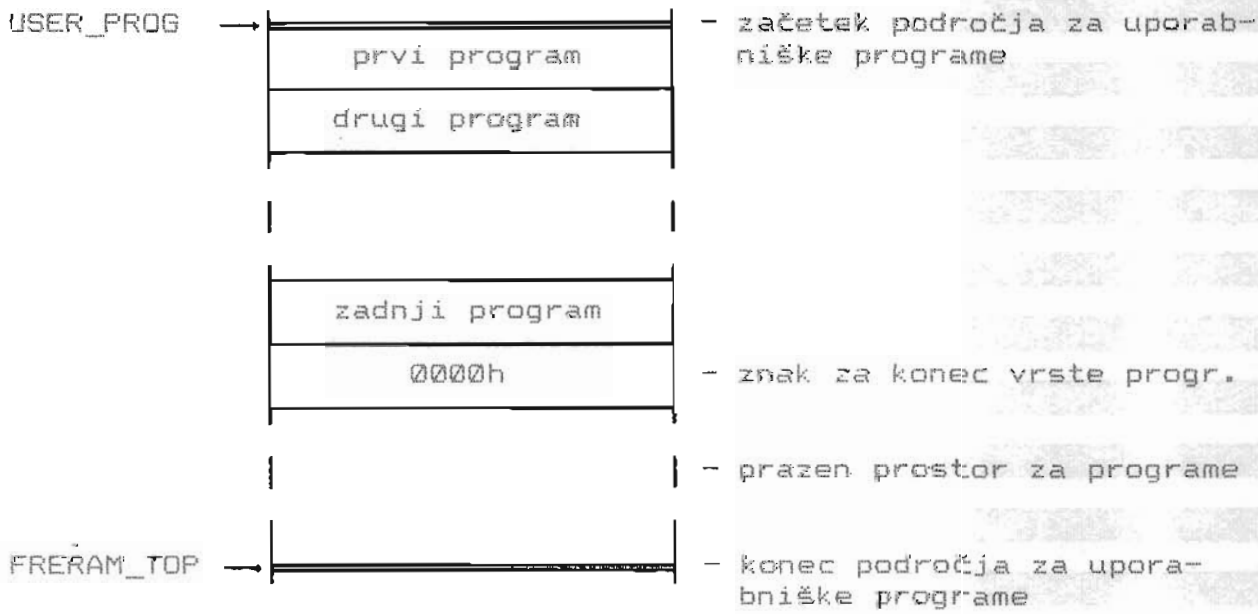
Uporabniške programe, ki so v RAMu, lahko vidimo v podrežimu PROGram HANDLE. Nad njimi je možno izvajati razne operacije (brisanje, preimenovanje, prepisovanje itd). Te programe lahko prenašamo na zunanje medije ali z zunanjih medijev preko RS232 povezave (v podrežimu DATA TRANSFer).

Uporabniške programe lahko editiramo v podrežimu CURRent PROGram

OBLIKA UPORABNIŠKIH PROGRAMOV V RAM-u

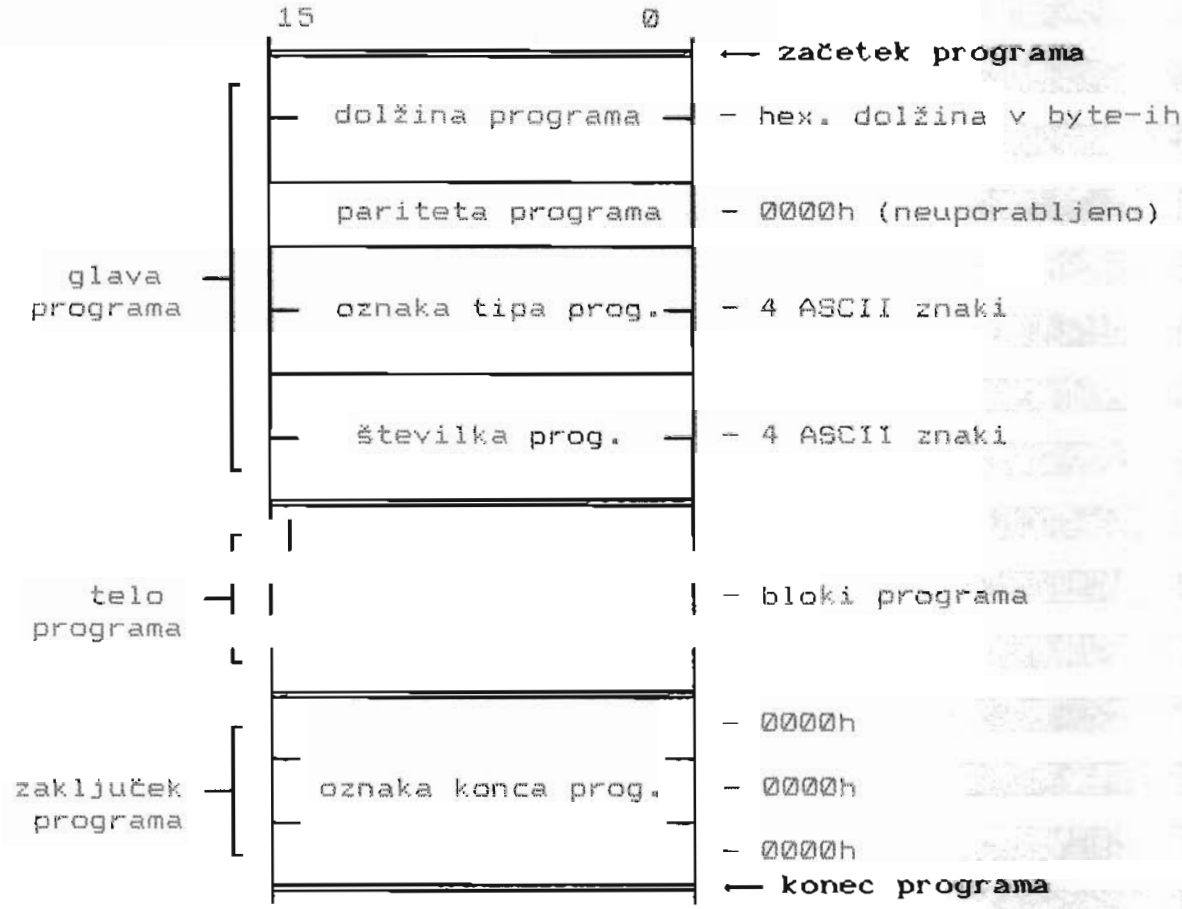
Uporabniški programi se nahajajo v uporabniškem področju RAM-a. Konec vrste programov označuje 0000h vrednost za koncem zadnjega programa.

Slika uporabniškega področja rama :



Vsak program v RAM-u je sestavljen iz:
- glave programa,
- blokov, ki tvorijo telo programa,
- zaključka programa.

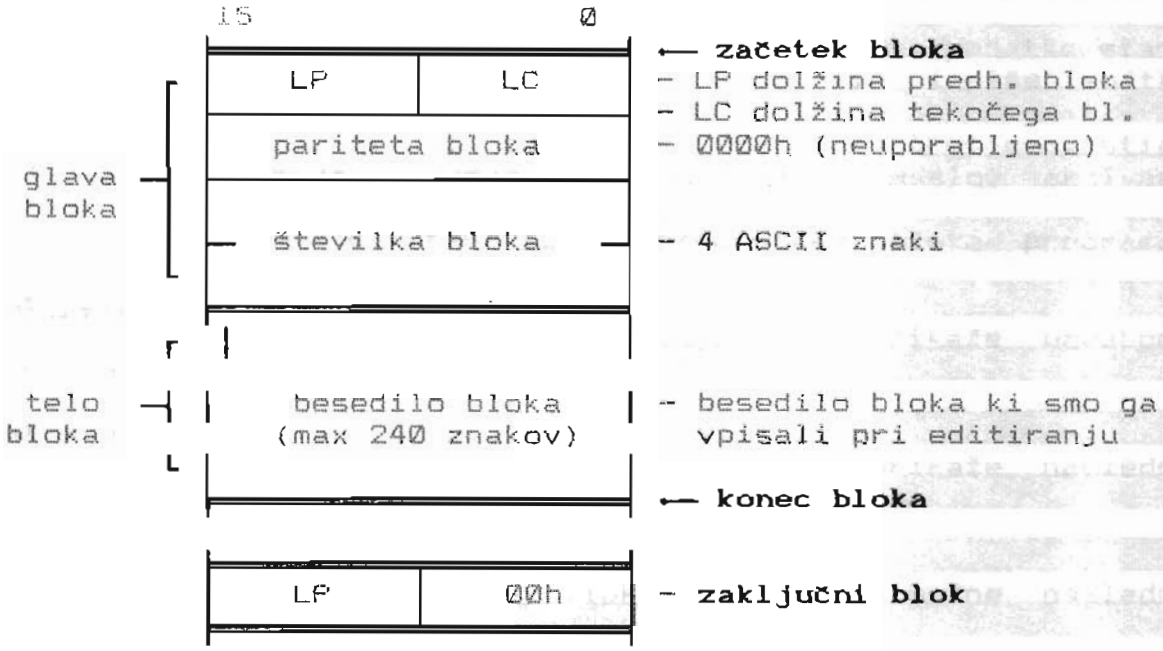
Slika programa v RAM-u:



Vsak blok programa v RAM-u je sestavljen iz:
- glave bloka,
- telesa bloka,

Za zadnjim blokom programa obstaja vedno še t.i. zaključni blok, ki je sestavljen le iz dolžine predhodnega bloka in 00h vrednosti (za dolžino naslednjega bloka).

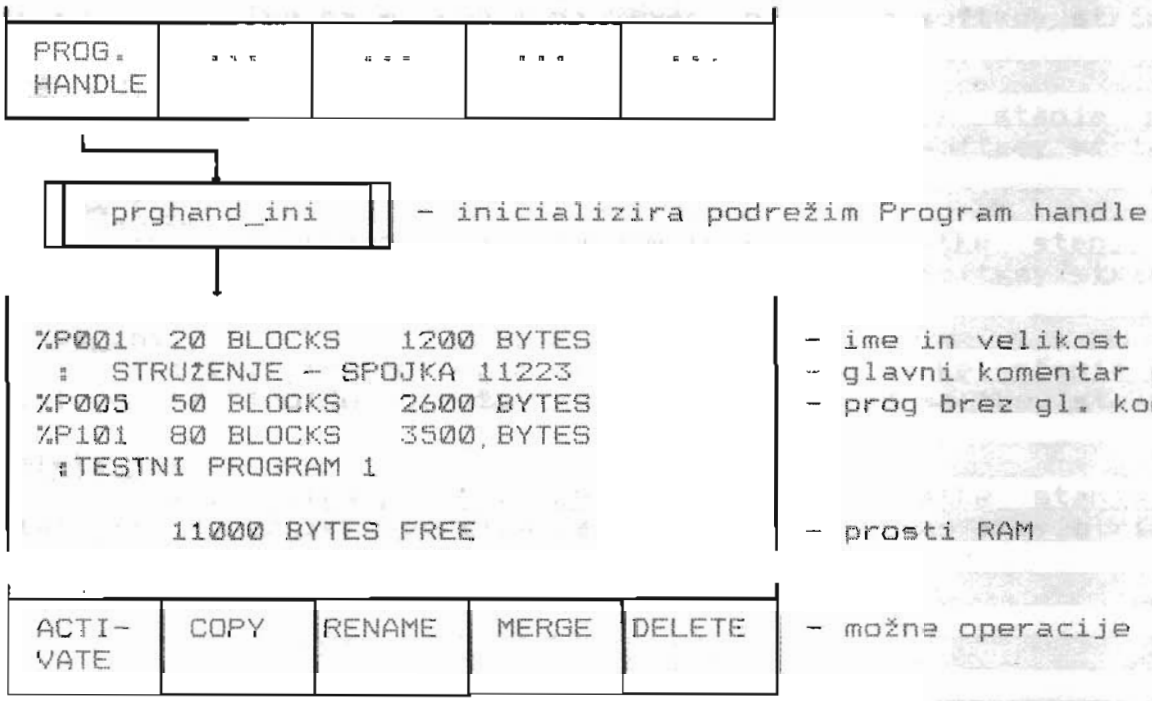
Slika bloka v RAM-u (glej naslednjo stran):



Operacije nad uporabniškimi programi

Podrežim PROGram HANDLE nam omogoča operacije nad uporabniškimi programi (UP). Ko pridemo v ta podrežim, se na ekran avtomatično izpiše prva stran liste UP. Če je v spominu več programov kot jih gre na ekran, lahko s pritiskom na tipki panoja za listanje strani gledamo poljubno stran liste programov. Za vsak program se na ekranu izpiše njegovo ime, dolžina v byte-ih in blokih ter glavni komentar programa (t.j. komentar iz prve vrstice programa). Na zadnji strani liste se izpiše število praznih byte-ov pomnilnika.

Primer izbire Program handle podrežima:



prghand_ini:
- inicializira podrežim Program handle (shrani podatke stanja na state_stack, izpiše podrežim na ekran in izpiše softkey stringe),
- na mesto za izpis 2. podrežima (začetek tretje vrstice) izpiše ime aktivnega programa (če obstaja). Naslov aktivnega programa vsebuje kazalec **CURPRG**.
- v delovno polje ekrana izpiše prvo stran liste programov.

plist_pgup:
Če nismo na prvi strani liste programov, prikaže predhodno stran liste.

plist_pgdwn:
Če nismo na zadnji strani liste programov, prikaže naslednjo stran liste.

Kot je razvidno iz softkey tipk, so nad UP možne naslednje operacije:
- aktiviranje programa,
- prepisaovanje programa,
- preimenovanje programa,
- združevanje programov,
- brisanje progrov.

Pritisk na eno od softkey tipk povzroči odprtje novega podrežima. V tem podrežimu napišemo imena programov nad katerimi želimo operacijo. Pritisk na tipko ENTER aktivira rutino, ki opravi ustrezno operacijo in nas vrne v prejšnji režim (Program handle).

Softkey tipkam v režimu Program handle pripadajo naslednje rutine:

activate_ini:
- inicializira podrežim activate (shrani podatke stanja na state_stack, izpiše podrežim na ekran in izpiše softkey stringe).

copy_ini:
- inicializira podrežim copy (shrani podatke stanja na state_stack, izpiše podrežim na ekran in izpiše softkey stringe).

rename_ini:
- inicializira podrežim rename (shrani podatke stanja na state_stack, izpiše podrežim na ekran in izpiše softkey stringe).

merge_ini:
- inicializira podrežim merge (shrani podatke stanja na state_stack, izpiše podrežim na ekran in izpiše softkey stringe).

delete_ini:
- inicializira podrežim delete (shrani podatke stanja na state_stack, izpiše podrežim na ekran in izpiše softkey stringe).

V vsakem od teh podrežimov se ob pritisku tipke ENTER aktivira ustrezna rutina. Te rutine so:

activatep:

- Če program s tem imenom obstaja, zapiše njegov naslov v globalni kazalec aktivnega programa (**CURPRG**), naslov prvega bloka pa v kazalec 1. bloka (**CURBLK_ED**).
- Če program še ne obstaja, ga odpre v prostem delu rama ter postavi oba kazalca (kot zgoraj).
- aktivira inicaializacijsko rutino predhodnega stanja (prghnd_ini) t.j. prehod v predhodno stanje.

copyp:

Izbrani program (source) prepiše v program z drugim imenom (destination). Če destination program že obstaja, ga na uporabnikovo željo prepiše. Če za prepis programa ni dovolj prostora, to javi. Po uspešno prepisanem programu se vrne v predhodno stanje.

renamep:

Izbrani program (source) preimenuje v nov program (destination). Če destination program že obstaja, ga na uporabnikovo željo uniči. Po uspešnem preimenovanju se vrne v predhodno stanje.

mergep:

Enega ali več programov (source programi) združi v program z izbranim imenom (destination). Če destination program že obstaja, ga na uporabnikovo željo unuči. Če za združitev programov ni dovolj prostora, to javi. Po uspešni združitvi se vrne v predhodno stanje.

delp:

Zbriše željene programe. Za brisanje vsakega programa zahteva dodatno potrditev. Program briše tako, da premakne programe, ki so za njim, čez njega. Po uspešnem brisanju se vrne v predhodno stanje.

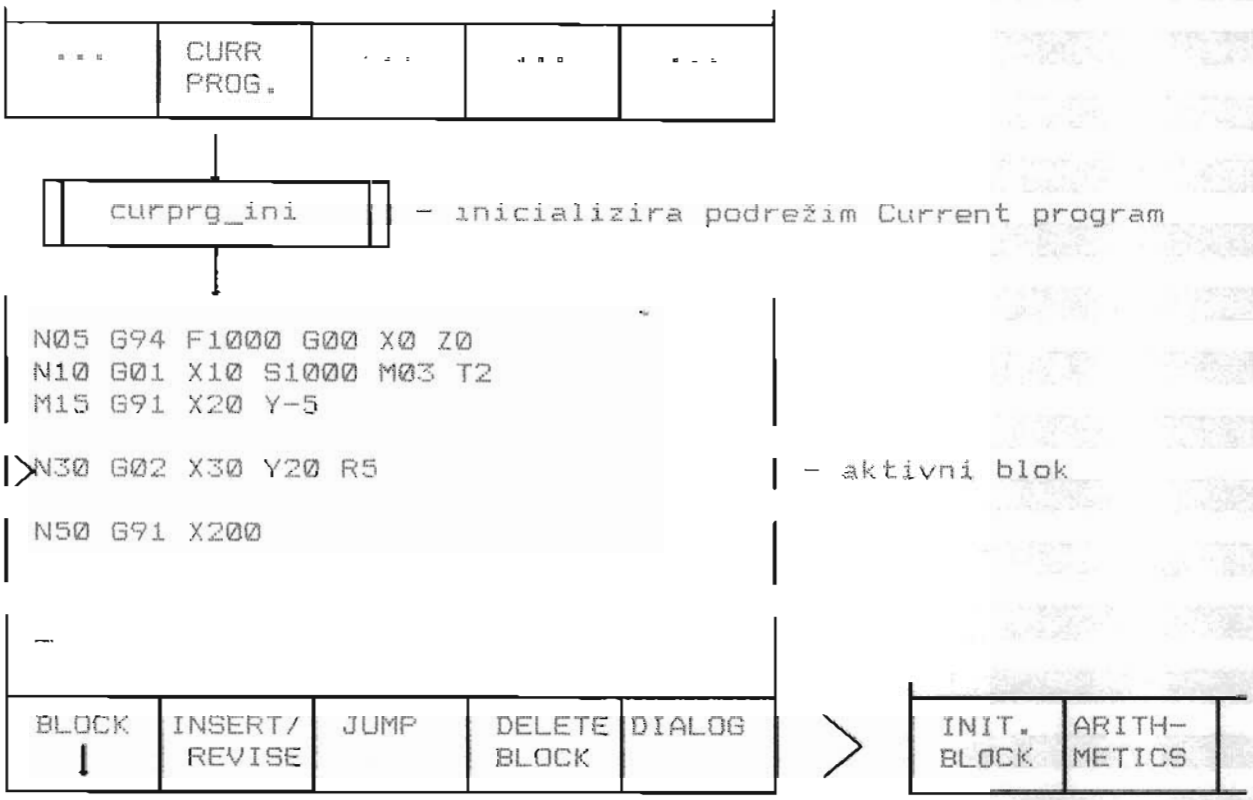
EDITIRANJE UPORABNIŠKIH PROGRAMOV (NC EDITOR)

Podrežimu CURRent PROGram nam omogoča editiranje uporabniških programov. Program, ki ga želimo editirati, moramo najprej aktivirati v podrežimu Program handle. Nato gremo v podrežim Current program, kjer ga lahko editiramo. Ob prihodu v ta podrežim, se na ekran avtomatično izpiše prva stran aktivnega programa ter softkey tipke, ki nam pomagajo pri editiranju. S pomočjo tipk panela lahko listamo program naprej in nazaj po vrsticah in straneh.

Obstajajo štirje načini editiranja in sicer:

- insert za vrivanje novega bloka,
- revise za popravljanje obstoječega bloka programa,
- auto increment, ki po vnosu bloka v program avtomatično izpiše številko novega bloka na začetek editorskega polja,
- normal, ki ne izpisuje številk bloka avtomatično v edit polje.

Primer prehoda v podrežim Current program:



curprg_ini:

- inicializira podrežim Current program (shrani podatke stanja na state_stack, izpiše podrežim na ekran in izpiše softkey stringe),
- v delovno polje ekrana izpiše prvo stran aktivnega programa in izpiše puščico aktivnega bloka na začetek 9. vrstice ekrana (to je sredina delovnega polja). Če smo v insert načinu editiranja, je vrstica na katero kaže puščica prazna.

curprg_up:

Če nismo v prvi vrstici programa, skoči puščica na predhodni blok aktivnega programa. Če smo bili ob tem v prvi vrsti delovnega polja ekrana, se prikaz premakne navzgor (scroll down). Če smo v insert načinu editiranja, je aktivna vrstica prazna.

curprg_dwn:

Če nismo v zadnji vrstici programa, skoči puščica na naslednji blok aktivnega programa. Če smo bili ob tem v zadnji vrsti delovnega polja ekrana, se prikaz premakne navzdol (scroll up). Če smo v insert načinu editiranja, je aktivna vrstica prazna.

curprg_pgup:

Če nismo na prvi strani programa, prikaže predhodno stran programa. Aktivni blok je blok v srednji vrstici delovnega polja. Nanj kaže puščica aktivne vrstice. Če smo v insert načinu editiranja, je aktivna vrstica prazna.

curprg_pgdwn:

Če nismo na zadnji strani programa, prikaže naslednjo stran programa. Aktivni blok je blok v srednji vrstici delovnega polja. Če smo v insert načinu editiranja, je aktivna vrstica prazna.

Načini editiranja, editorske funkcije in dialog so nam dostopni preko softkey tipk. Nov blok programa lahko tvorimo na dva načina:

1). z alfanumeričnimi tipkami panela in/ali s pomočjo dialoga napišemo čisto nov blok v editorskem polju. Tega lahko nato vrinemo na željeno mesto v programu (insert način) ali pa z njim prepišemo željeni blok programa (revise način),

2). v revise načinu prenesemo s pritiskom na BLOCK ↓ softkey tipko željeni blok iz programa v editorsko polje. Ta blok nato ustrezno predelamo in ga zapišemo v program (kot zgoraj).

nced_ins:

Aktivira se ob pritisku na INSERT/REVISE softkey tipko. Izvrši prehod iz revise v insert način ali obratno V globalni spremenljivki **ED_MODE** spremeni ustrezni bit. Če smo bili v revise načinu, vrine na mesto puščice prazno vrstico, če pa smo bili v insert načinu, prazno vrstico zbriše.

down_block:

Aktivira se ob pritisku na BLOCK ↓ softkey tipko. Če smo v revise načinu, prenese aktivni blok programa iz RAMa v editorski vmesnik. Če smo v insert načinu ni akcije.

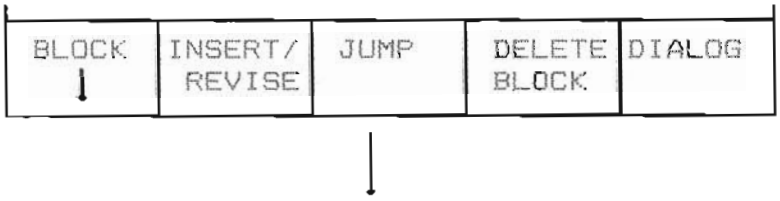
ncedit:

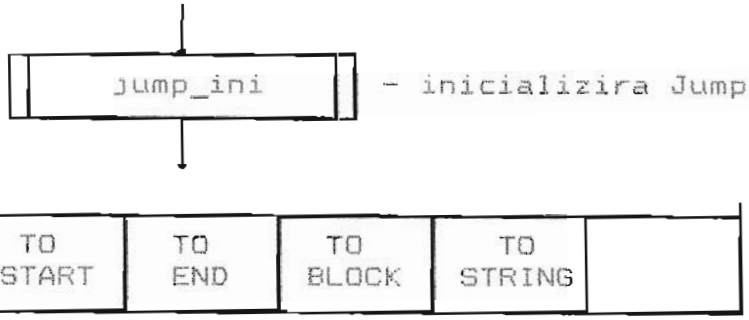
Aktivira se ob pritisku tipke ENTER v podrežimu Current program. Blok zapisan v editorskem polju predela v obliko za zapis v RAM. Če smo v revise načinu, prepiše trenutno aktivni blok z novim, če pa smo v insert načinu, ga vrine na željeno mesto. Če je v Setting data postavljen ustrezni flag, ne zbriše editorskega vmesnika ,ampak le premakne kursor na začetek editorskega polja. Če smo v auto increment načinu, poveča številko bloka in jo zapiše na začetek editorskega vmesnika.

del_block:

Aktivira se ob pritisku na DELETE softkey tipko. Če smo v revise načinu, zahteva od uporabnika potrditev za brisanje. Ko dobi potrditev, briše blok na katerega kaže puščica v delovnem polju.

Pritisk na JUMP softkey tipko odpre podrežim, ki omogoča premike po programu in iskanje stringa v programu. Primer prehoda v JUMP podrežim:





- inicializira Jump podrežim

jump_ini:
- inicializira JUMP podrežim (shrani podatke stanja na state_stack, izpiše podrežim na ekran in izpiše softkey stringe).

Softkey tipke tega podrežima nudijo naslednje možnosti:
- skok na začetek programa,
- skok na konec programa,
- skok na blok z določeno številko,
- skok na določen string v programu.

to_pstart:
Izpiše prvo strani programa, aktvira prvi blok programa in nanj usmeri puščico.

to_pend:
Izpiše zadnjo stran programa, aktvira zadnji blok programa in nanj usmeri puščico.

to_block:
Povzroči izpis strani programa, na kateri se izbrani blok nahaja na sredini delovnega polja ekrana, ga aktivira in nanj usmeri puščico.

to_string:
Poišče izbrani string v programu od tekočega bloka naprej. Če ga najde, izpiše stran programa na kateri se blok, ki vsebuje iskani string, nahaja na sredini delovnega polja ekrana. Blok aktivira in nanj usmeri puščico.

Za pomoč uporabniku pri programiranju služita podrežima Dialog in Arithmetics Current program režima.

Dialog podrežim preko softkey tipk in Choose dialogov omogoča izbiro različnih NC funkcij kot npr.:
- gibalnih funkcij,
- M funkcij,
- obstoječih podprogramov in fiksnih ciklov,
- Zero offset-ov,
- korekcij orodja itd.

Arithmetics podrežim odpre Choose dialog za izbiro matematičnih funkcij (SIN, COS, SQTR, ...).
V obeh podrežimih uporabnik izbira funkcije, ki se mu avtomatično vpišejo na mesto kursorja v editorskem polju in mu jih ni treba vnašati preko alfanumeričnih tipk.

9. PRILOGA A: RUTINE POGLAVIJ 1-4 V JEZIKU C

AVTOMAT STANJ

```
/**** Stuktura stanja (state_strc) ****/
struct state_strc
{
     int (*ini_rut)();        /* kazalec na inic. rutino */
     struct keyact_strc *keyact;   /* kazalec na str. akt tipk in
                                      akcijskih rutin */
     struct sks_strc *sks;    /* kazalec na str.  softkey  stringov
                                      in rutin */
};

/**** Struktura tipke in akcije (keyact_strc) ****/

struct keyact_strc
{
     int rkey;         /* redukcijska koda - word! */
     int (*act)();     /* kazalec na akcijsko rutino */
};


/**** Struktura softkey stringa in rutine (sks_strc) ****/
struct sks_strc
{
     char str[20];     /* string softkey-a in '\0' */
     int (*func)();    /* kazalec na rutino */
};



key_pars()
{
     struct keyact_strc *ptkeyact;       /* kazalec na keact str. */

     ptkeyact =  curr_state->keyact;     /* kaze  na  prvo  str.  */


     while(pfkeyact->rkey)
     {
          if(pfkeyact->rkey == rkey)
          {
               pfkeyact->act();     /* izvedemo akcijo */
               return;
          }
          pfkeyact++; /* inkrementiramo */
     }
}
```

```
serve_sks()
{
     int i;
     struct sks_strc *ptsks;        /* kazalec na sks strukturo */

     ptsks =  curr_state->sks;       /* kaze  na  prvo  str.  */
    /** izracun offseta softkey. tipke **/
     i = curr_fks * 5  + keyb - SKS_OFFS ;
     ptsks += i;     /* kazalec na pravo strukturo */
     if(ptsks->func)      /* polna tipka ? */
          ptsks->func();      /* da, izvedemo rutino */
}


roll_sks()
{
     int i;
     struct sks_strc *ptsks;        /* kazalec na sks strukturo */

     ptsks =  curr_state->sks;       /* kaze  na  prvo  str.  */

     curr_sks++;     /* inkrementiranje stevca sks strani */
     i = curr_sks * 5 -1       /* ofset aktivne sks strani */
     ptsks += i;          /* inkrementiramo kazalec */

     /* test konca  strani */
     if(!ptstr->str[0])      /* smo izven ? */
          curr_sks = 0;         /* da, gremo na 1. stran */
     sks_disp();          /* izpis nove strani */
}

to_pstate()
{
     struct state_strc *ptsstr;     /* kazalec na str. stanja */

    /* dekrementiramo indeks tekocega stanja */
     cstate -= 2;
     if(cstate < 0)
          cstate = 0;

     /* dekementiramo kazalec strukture stanja */
     curr_state -= 2;

     if(curr_state < state_stack)
     {          /* gremo v glavni rezim */
          curr_state = state_stack;
          (curr_state)->ini_rut();     /* izvrsimo inic. rutino */
          return;          /* ker smo bili v osnovnem stanju */
     }
     else
     {
          ptsstr = curr_state;
          ptsstr ++;
          ptsstr->ini_rut();     /* izvrsimo inic. rutino */
     }
}
```

```
sw_mstate()
{
      /* dolocimo pritisnjeno tipko */
      switch(key)
      {
            case(MANK):          /* zelimo v manual rezim */
                  dmode = 1;
                  break;
            case(DIOK):          /* zelimo v data-i/o rezim */
                  dmode = 2;
                  break;
            case(AUTK):          /* zelimo v automatic rezim */
                  dmode = 3;
                  break;
            case(BBBK):          /* zelimo v block by block rezim */
                  dmode = 7;
                  break;
      }
      if(waitsw_indx == -1)
            waitsw_indx = act_perr(wait_sw, 1);     /* aktiviramo rutino *

}


waitsw()
{
      int i;

      if(ECMODE)
            return;    /* preklop ni dovoljen - gremo ven */

      MODE = dmode;
      acmode = MODE;       /* !! preveri ce je isto */

      /* izracunamo indeks kazalca inicializacijske rutine */
      if(dmode == 7)
            i = 4;
      else
            i = dmode;
      i--;

      *ini_ruts[i]();      /* izvedemo inicializacijsko rutino */
      dact_perr(waitsw_indx);  /* se deaktiviramo */
      waitsw_indx = -1;    /* znak sw_mstate rutini */




}
```

P E R I O D I Č N E R U T I N E

```
/**** Struktura perr. rutine v vrsti (perr_strc) ****/

struct perr_strc
{
     union w2b type; /* aktivna (lsb = 1), perm. (msbyte = 1) */
     int (*perr)(); /* kazalec na per. rutino */
}

union w2b
{
     int        word;
     short      byte[2];
};


perex()
{
     int i;

     for(i=0; i<10; i++)
          if(perr_area[i].w2b.byte[0])
               *perr_area[i].perr();     /* jo izvedemo */

}


ini_perr()
{
     int i;

     for(i=0; i<10; i++)
     {
          perr_area[i].w2b.word = 0;     /* neakt. in neperm */
          perr_area[i].perr = 0;         /* kazalec na 0 */
     }
}


int act_perr( int (*kperr)(), int tip )
{
     int i;

     for(i=0; i<10; i++)
          if(!perr_area[i].type.byte[1])
          {                         /* prazno mesto */
               perr_area[i].perr = kperr;     /* kazalec na rut. */
               perr_area[i].type.byte[0] = tip;
               perr_area[i].type.byte[1] = 1;  /* aktivna rut. */
               return i;            /* vrnemo index rutine */
          }

     return -1;      /* ni prostora - napaka */
```

```
}
int dact_perr( int index)
{
     if(perr_area[index].w2b.byte[0])
     {
          perr_area[i].w2b.cel = 0;  /* deaktiviramo */
          return 0;
     }
     return 1;       /* taka perr. ne obstaja */
}


clr_perr()
{
   int i;

   for(i=0; i<10; i++)
        if(perr_area[i].w2b.byte[1] && !perr_area[i].w2b.byte[0])
            perr_area[i].w2b.word = 0; /* neakt. in neperm. */

}
```

DIAGNOSTIKA

```
/**** al_llist (alarm linked list) ****/

struct al_strc {
          short nxt;      /* index naslednjega sporo~ila */
          al[41];         /* string sporo~ila (40 znakov + 0) */
}al_llist[32];

/**** al_cque (alarm circular queue) ****/

int  al_cque[10];
int  cq_spos,        /* circular queue store position */
     cq_rpos;        /* circular queue retreive position */


ini_allist()
{
     int i;

     /* al_llist */
     for(i=0; i<32; i++)
          al_llist[i].al[0] = '\0';    /* izpraznimo tabelo */
     mimp_al = -1;          /* ni sporocila */

     /* al_cque */
     cq_spos = 0;
     cq_rpos = 0;

     al_setting = 0;      /* inic. flag ki pove disp_al-u, da se
                             novo sporocilo ravno vpisuje v vrsto */
```

```
}
int set_al(char *new_al)
{
     int  indx;      /* za index rutine */


     al_setting++;  /* disp_al ne sme iti v  urejanje,  dokler  ne
                       bomo vpisali novega alarma */

    /* poiscemo prvo prazno mesto v tabeli */
     for(indx=0; indx<32 && al_llist[indx].al[0]; indx++) ;

     if(indx == 32)
     {
          al_setting--;  /* sprostimo disp_al */
          return -1;     /* vrsta je polna - ni uvrstitve */
     }

     /** prepisemo alarm v tabelo **/
     strncpy(al_llist[indx].al, new_al, 40);
     /** konec stringa **/
     al_llist[indx].al[40] = 0;

     /*** test al_cque ***/
     if(cq_spos+1 == cq_rpos || (cq_spos+1 == CQMAX && !cq_rpos))
     {          /* vmesnik je polen */
          al_llist[indx].al[0] = 0;  /* sprostimo mesto v vrsti */
          al_setting--;  /* sprostimo disp_al */
          return -1;     /* javimo da ni uvrstitve */
     }

     al_cque[spos++] = indx;  /* zapisemo indeks v vrsto */
     if(cq_spos == CQMAX)
          cq_spos = 0;    /* na zacetek vmesnika - krog */
     al_setting --;       /*** z nase  strani  je  zadeva  urejena,
                               obstaja pa lahko drug modul ki  je  v
                               tej rutini (al_setting <> 0)    ***/

     return indx;    /* index sporocila ki je v vrsti cakajocih */

}



disp_al()
{
     int  indx,      /* za index rutine */
          old,       /* index predhodnega alarma */
          curr,      /* tekoci index */
          full;      /* flag za polno tabelo */

    indx = -1;       /* inic. za osvezitev prikaza v Diag rezimu */
     if(!al_setting)
    {          /* uvrscanje je dovoljeno */
          while(cq_rpos  !=  cq_spos)
          {          /* dokler ne izpraznemo vmesnika */
```

```
if(cq_rpos+1 == CQMAX)    /* ce smo na koncu obrnemo */
{
     cq_rpos = 0;
     continue;
}

/* vzamemo tekoci index iz vmesnika */
indx = al_cque[cq_rpos++];

if(!al_llist[indx].al[0])
     continue;       /* zbrisan pred uvrstitvijo */

/* ce je al_llist prazen */
if(mimp_al == -1)
{
     new_1al = 1;    /* znak, da je nov 1. alarm */
     mimp_al = indx;      /* indeks 1. alarma */
     al_llist[indx].nxt = -1;  /* je tudi zadnji al. */
     continue;        /* na uvrscanje naslednjega */
}
else       /* ni prazna tabela */
{
     old = mimp_al;       /* zacnemo na zacetku */
     curr = old;

     /* glavna zanka ki poisce mesto alarma */
     do
     {
          if (bigger_al(al_llist[indx].al, al_llist[curr
].al))

          {    /* nov al. je vaznejsi od tekocega */
               if(curr == mimp_al)
               {          /* je najvaznejsi alarm */
                    /* popravimo index 1. al. */
                    mimp_al = indx;
                    new_1al = 1;      /* nov 1. al. */
               }
               else
                    /* prejsnji mora kazati na novega */
                    al_llist[old].nxt = indx;

               /** postavimo kazalec na naslednji al. **


               al_llist[indx].nxt = curr;
               continue;  /* na nasl. cakajoci al. */
          }
          old = curr;     /* inkrementiramo */
          /*** curr. je naslednji al. po prioriteti ***
          curr = al_llist[curr].nxt;

     }while(curr != -1)    /* do konca vrste alarmov */

     /* al. je zadnji po prioriteti */
     al_llist[old].nxt = indx;      /* kaz. prejsnjega */
     al_llist[indx].nxt = -1; /* je zadnji v vrsti */

}
} /* konec while stavka */
```

```
        }

        /*** izpis najpomembnejsega sporcila, ce je nov */
        if(new_1al)     /* sprememba prvega alarma ? */
        {
             if(mimp_al != -1)    /* da. nov alarm ? */
                  puts_xy(atrib, x, y, al_llist[mimp_al].al); /* da */
             else       /* prazna vrsta */
                  puts_xy(atrib, x, y, Empty_string); /* define !*/
             new_1al = 0;    /* podremo flag */
        }

        /** ce smo v diag. in je prislo do spremembe vrste */
        if(curr_mode == DIAG && indx == -1)
             outp_diag();

}


int  bigger_al(short *al1, short *al2)
{
     int i;

     if(*al1 & 0xF < *al2 & 0xF)
          return 1;       /* je vecji po prioriteti */

     for(i=1; i<6; i++)
          if(*al1 < *al2)
               return 1;       /* je vecji po prioriteti */

     return 0;       /* manjsi ali enak */
}
```

```
int clr_al(int indx)
{
     register int curr, old;

     curr = mimp_al       /* inicializacija kazalca */
     old = curr;
     /*** iscemo alarm z indx do konca vrste ***/
     While(curr != indx && curr != -1)
     {
          old = curr;
          curr = al_llist[curr].nxt;     /* naslednji alarm */
     }
     if(curr = -1)
          return 0;       /* nismo ga nasli */
     else if(curr == mimp_al)
     {          /** je prvi v vrsti **/
          new_1al = 1;    /* znak disp_al-u, da je nov 1. alarm */
          mimp_al = al_llist[curr].nxt;  /* indeks 1. alarma */
     }
     else       /* ni prvi al. v vrsti */
          al_llist[old].nxt = al_llist[curr].nxt;
    al_llist[curr].al[0] = 0;       /* znak za prazen string */
     /** ce smo v diag. osvezimo prikaz */
     if(curr_mode == DIAG )
          outp_diag();
     return 0;
}

clral_nep()
{
     register int curr, old;

     curr = mimp_al       /* inicializacija kazalca */
     old = curr;
     /*** iscemo neperm. alarm do konca vrste ***/
     While(curr != -1)
     {
          if(!al_llist[curr].al[1] & 0x80)
          {          /* nepermantno sporocilo */
               al_llist[curr].al[0] = 0;  /* brisemo al. string */
               if(curr == mimp_al)
               {          /** je prvi v vrsti **/
                    new_1al = 1;    /* znak disp_al-u, nov 1. alarm */
                    /*** indeks 1. alarma  */
                    mimp_al   =  al_llist[curr].nxt;
               }
               else       /* ni prvi al. v vrsti */
                    al_llist[old].nxt = al_llist[curr].nxt;
          }
          else       /* permanentno sporocilo */
               old = curr;          /* old je lahko le perm. al. */
          curr = al_llist[curr].nxt;
     }
     /** ce smo v diagnosticnem podrezimu osvezimo ekran */
     if(curr_mode == DIAG)
          outp_diag();
```

```
}
clral_usrn(int user_numb)
{
     register int curr, old;
     int  numb,      /* buffer za stevilko userja */
          outp;      /* za shranitev return vrednosti */

     outp = 1;       /* inicializiramo na napako */
     curr = mimp_al       /* inicializacija kazalca */
     old = curr;
     /*** iscemo neperm. alarm do konca vrste ***/
     While(curr != -1)
     {
          numb = -1;      /* za detekcijo napake pretvorbe */
          /*** ascii string dolzine 2 z 0 decimalkami v int.,
               pri nep. sporocilih prvi znak ni ascii -> napaka */
          ascbin_w(al_llist[curr].al+1, 2, 0, &numb);
          if(numb == user_numb)
          {          /* pravi user in nepermantno sporocilo */
               al_llist[curr].al[0] = 0;  /* brisemo al. string */
               if(curr == mimp_al)
               {          /** je prvi v vrsti **/
                    new_ial = 1;   /* znak disp_al-u, nov 1. alarm */
                    mimp_al = al_llist[curr].nxt;  /* indx 1.al. */
               }
               else       /* ni prvi al. v vrsti */
                    al_llist[old].nxt = al_llist[curr].nxt;
               outp = 0;       /* zbrisali smo vsaj enega */
          }
          else       /* permanentno sporocilo */
               old = curr;          /* old je lahko le perm. al. */
          curr = al_llist[curr].nxt;
     }
     /** ce smo v diagnosticnem podrezimu osvezimo ekran */
     if(curr_mode == DIAG)
          outp_diag();

     return outp;
}
```

```
clral_scope(char *strn, int scope)
{
     register int curr, old;
     int  numb,      /* buffer za stevilko userja */
          outp;      /* za shranitev return vrednosti */

     outp = 1;       /* inicializiramo na napako */
     if(scope < 2)
          return outp;    /* napaka */
     else if(scope > 5)
          scope =5;       /* popravimo scope */
     curr = mimp_al       /* inicializacija kazalca */
     old = curr;
     /*** iscemo neperm. alarm do konca vrste ***/
     While(curr != -1)
     {
          numb = -1;      /* za detekcijo napake pretvorbe */
          /*** primerjava prvih scope znakov stringov ***/
         if(!strncmp(al_llist[curr].al+1, strn, scope))
          {          /* znaki se ujemajo */
               al_llist[curr].al[0] = 0;  /* brisemo al. string */
               if(curr == mimp_al)
               {          /** je prvi v vrsti **/
                    new_ial = 1;    /* znak disp_al-u, nov 1. alarm */
                    mimp_al = al_llist[curr].nxt;  /* indx 1.al. */
               }
               else       /* ni prvi al. v vrsti */
                    al_llist[old].nxt = al_llist[curr].nxt;
               outp = 0;       /* zbrisali smo vsaj enega */
          }
          else       /* permanentno sporocilo */
               old = curr;          /* old je lahko le perm. al. */
          curr = al_llist[curr].nxt;
     }
     /** ce smo v diagnosticnem podrezimu osvezimo ekran */
     if(curr_mode == DIAG)
          outp_diag();

     return outp;
}
```

EDITORSKI VMESNIK

```
char      editbuf[240];  /* editorski vmesnik */
int  curchr,   /* odmik tekoje pozicije od zacetka vmesnika */
    lastchr;   /*  odmi zadnjega znaka od zacetka vmesnika */

clr_eb()
{
     int i;
     for(i=0; i<=MAXEB; i++)
          editbuf[i] = ' ';          /* presledek */
     /* inicializacija stevcev */
     curchr = 0;
     lastchr = 0;
     first_seen = 0;  /* izrez prikaza ed */
     last_seen = 77;

     disp_eb();      /* osvezimo prikaz */
     not_clreb = 0;       /* glob. flag,  ki  pove,  da  je  po  CR
                             prva tipka editorska,  zato  se  eb.
                             ne brise */
}

edit_eb()
{
     switch(rkeyb)        /* pogledamo redukcijsko kodo tipke */
     {
          case EDIT :     /* editorska tipka */
               switch(keyb)   /* pogledamo norm. kodo tipke */
               {
                    case LEFT :     /* za en znak levo */
                         lefch_eb();
               putsn_xy(attr,editbuf+first_seen+39,17,39);
                         break;
                    case RIGHT :    /* za besedo v desno */
                         rghtwrd_eb();
                         break;
                    case INSERT :  /* insert. oz revise nacin */
                         insmod_eb();
                         break
                    case DELETE :  /* brisanje znaka ali besede */
                         delcw_eb();
                         break;
                    case SHDEL :    /* shift delete - brisanje eb.*/
                         clr_eb();
                         break;
               }
               break;
          case PUNCT :    /* locilo */
          case ALFA :     /* crka */
          case NUM :      /* cifra */
               putc_eb();      /* zapis znaka */
               break;

     }
     disp_eb();      /* osvezitev prikaza eb. na ekranu */
```

```
}

leftch_eb()
{
      if(curchr)
            curchr--;
      if(curchr<first_seen)
      {
            first_seen--;
            last_seen--;
      }

      not_clreb = 1;       /* postavimo flag za ohranitev eb. */
}

rghtwrd_eb()
{

      /* gremo do konca tekoce besede */
      while(curchr < lastchr && editbuf[curchr] != ' ')
            curchr++;
      /* gremo do zacetka desne besede */
      while(curchr < lastchr && editbuf[curchr] == ' ')
            curchr++;
      /* ce je bila zadnja beseda */
      if(curchr == lastchr && lastchr < MAXEB)
      {
            curchr++;
            lastchr++;
            editbuf[curchr] = ' ';    /* lahko so ostale smeti */
      }
      if(curchr>last_seen)
      {
            last_seen = curchr;
            first_seen = curchr - 77;
      }

      not_clreb = 1;       /* postavimo flag za ohranitev eb. */
}
```

```
insmod_eb()
{
      int nchr;       /* vmesnik za stevilo prepisanih znakov */

      if(curchr == MAXEB)      /* smo na koncu ed. polja */
            return;
      if(insert)      /* insert -> revise */
      {
          insert = 0;
          /* premik za znak v levo - zbrisemo presledek */
          memmove(editbuf+curchr,editbuf+curchr+1,lastchar-curchr);
          lastchr--;       /* dolzina besedila se je zmanjsala */
      }
      else            /* revise -> insert */
      {
            insert = 1;
            if(lastchr < MAXEB)      /* se ni polno edit polje */
            {
                  /* stevilo znakov za prepis */
                  nchr = lastchr - curchr + 1;
                  lastchr++;      /* besedilo se bo podaljsalo */
            }
            else       /* edit polje je polno - zadnji znak izpade */
                  nch = lastchr - curchr;
            /* naredimo prostor za isert presledek */
            memmove(editbuf+curchr+1, editbuf+curchr, nchr);
            editbuf[curchr] = znak;  /* vpisemo presledek */
      }
      not_clreb = 1;       /* postavimo flag za ohranitev eb. */
}
```

```
delcw_eb()
{
     int  i,
          nchr;     /* vmesnik za stevilo znakov za prepis */

     if(editbuf[curchr]!=' '&& (!curch||editbuf[curchr-1]==' '))
     {    /* brisanje cele besede */

          i = curchr;     /* inic. stevca */

          /* pogledamo do konca tekoce besede */
          while(i<lastchr && editbuf[i] != ' ')
               i++;
          /* pogledamo do zacetka desne besede */
          while(i<lastchr && editbuf[i] == ' ')
               i++;

          nchr = i - curchr;  /* stevilo znakov za premik */
          /* premik za i znakov v levo/
          memmove(editbuf+curchr, editbuf+i, lastchr-i+1);
          lastchr = lastchr - nchr;   /* besedlo je krajse */

          if(curchr == lastchr)     /* bila je zadnja beseda */
          {
               editbuf[curchr] = ' ';    /* presledek */
          }
     }
     else if(!curchr && editbuf[curchr] = ' ' && insert)
          /* na zacetku na presledku in smo v revse nacinu */
          insmod_eb();     /* gremo v revise - presledek se brise */

     else
     {    /* brisanje predhodnega znaka ali ' 'na zacetku */
          if(curchr)
               curchr--;
          /* prepis v levo za znak */
         memmove(editbuf+curchr,editbuf+curchr+1,lastchar-curchr);
         lastchr--;       /* dolzina besedila se je zmanjsala */
          if(curchr<first_seen)
          {
               first_seen--;
               last_seen--;
          }

     }
     not_clreb = 1;       /* postavimo flag za ohranitev eb. */
}
```

```
putc_eb( char znak )
{
     int nchr;       /* vmesnik za stevilo prepisanih znakov */

     if(curchr == MAXEB)      /* smo na koncu polja - izhod */
          return;

     if(!curchr && !not_clreb)
          clr_eb();       /* brisemo editorski vmesnik in polje */

     if(insert)
     {
          if(lastchr < MAXEB)      /* se ni polno edit polje */
          {
               /* stevilo znakov za prepis */
               nchr = lastchr - curchr + 1;
               lastchr++;      /* besedilo se bo podaljsalo */
          }
          else       /* edit polje je polno */
               nch = lastchr - curchr;

          /* naredimo prostor za nov znak */
          memmove(editbuf+curchr+1, editbuf+curchr, nchr);
          editbuf[curchr] = znak;  /* prepisemo znak */
          curchr++;       /* kursor na naslednji znak */
     }
     else       /* revise nacin */
     {
          editbuf[curchr] = znak;  /* prepisemo znak */
          curchr++;       /* kursor na naslednji znak */
     }
     if(curchr>last_seen)
     {
          first_seen++;
          last_seen++;
     }

}
```

```
disp_eb()
{
     curs_off();     /* ugasnemo kursor - CRT rutina */

     goto_xy(1,16);
     if(first_seen)       /* levo je text */
          putc('>');
     else
          putc(' ');

     if(lastchr < (last_seen-first_seen)/2-1))
     {          /* besedilo gre v eno vrsto */
          putsn( editbuf+firs_seen, lastchr-first_seen);
          clreol();       /* brise do konca tekoce vrste */
          clrline(17);    /* mogoce je ostal text od prej */

     }
     else       /* besedilo v dveh vrstah */
     {
          /* izpis prve vrste */
          putsn( editbuf+first_seen, 39);

          if(lastchr>last_seen)     /* text na desni */
          {
               putsn_xy(attr,editbuf+first_seen+39,17,39);
               putc('>');
          }
          else
          {
               putsn_xy(attr,editbuf+first_seen+39,17,lastchr-
                         first_seen-39));
               clreol();       /* brise do konca tekoce vrste */
          }
     }
     /* postavimo kursor na tekoco pozicijo in ga prizgemo */
     curs_xy(curchr%39, 16 + int(curchr/39));
}
```