UNIVERZA V LJUBLJANI
FAKULTETA ZA ELEKTROTEHNIKO

UNIVERZITETNI ŠTUDIJ

DIPLOMSKO DELO

Prekinitve v večopravilnem operacijskem sistemu

Arpad Bűrmen

Mentor: doc. dr. Tadej Tuma

LJUBLJANA, 1999

ZAHVALA

Zahvaljujem se mentorju doc. dr. Tadeju Tumi, dipl. ing., ki mi je vseskozi pomagal s predlogi in ocenjeval napredek.

Prisrčna hvala tudi očetu Francu in mami Miroslavi, ki sta mi omogočila študij in me vseskozi podpirala.

Nenazadnje pa sem posebno zahvalo dolžan tudi bratu Miranu in mami Miroslavi, ki sta mi pomagala »loviti« (upam, da tudi »poloviti«) slovnične napake v besedilu, za kar sem jima nadvse hvaležen.

KAZALO

1 Povzetek ---------- *1*

2 Uvod ---------- *3*

3 Operacijski sistem RT68 ---------- *5*

3.1 Izvedba večopravilnosti v operacijskem sistemu RT68 ---------- **5**

3.2 Splošno o prekinitvah in sistemih v realnem času ---------- **6**

3.3 Statično dodeljevanje procesorskega časa ---------- **7**

3.4 Dinamično dodeljevanje procesorskega časa ---------- **10**

4 Nekaj podrobnosti o mikrokrmilniku M6803 ---------- *12*

4.1 Zahteve po prekinitvi ---------- **12**

4.2 Maskiranje prekinitev ---------- **13**

5 Operacijski sistem RT68 s podporo prekinitvam ---------- *18*

5.1 Odčitavanje prostotekočega števca ---------- **18**

5.2 Problem hkratnega dostopa do skupnih sredstev ---------- **18**

5.3 Onemogočanje prekinitev ---------- **20**

5.4 Omogočanje prekinitev ---------- **21**

5.5 Vzpostavitev začetnega stanja sistema ---------- **21**

5.6 Prekinitveni podprogrami ---------- **22**

5.7 Statično dodeljevanje procesorskega časa ---------- **23**

5.8 Dinamično dodeljevanje procesorskega časa ---------- **25**

5.9 Gnezdenje prekinitev in opravila, ki ne smejo biti prekinjena ---------- **27**

6 Primerjava obeh pristopov ---------- *28*

7 Sklep ---------- *31*

8 Priloga: preprečevanje hkratnega dostopa v operacijskem sistemu brez podpore prekinitvam ---------- *32*

9 Viri ---------- *33*

1 Povzetek

Operacijski sistem v realnem času, s katerim so opremljeni učni mikrokrmilniški sistemi RT68-01, RT68-02 in RT68-03, nima vgrajene podpore za izvajanje prekinitvenih podprogramov. Glede na dejstvo, da so prekinitve uporabljene praktično v vsakem mikrokrmilniškem sistemu, se pojavlja potreba po dograditvi operacijskega sistema, da bo ta zmožen obravnavati zahteve po prekinitvi, ki prihajajo iz zunanjih enot.

Mikrokrmilnik M6803, okoli katerega so učni mikrokrmilniški sistemi zgrajeni, ima vgrajeno strojno podporo prekinitvam. Na prvi pogled se zdi, da je potrebno samo omogočiti prekinitve. Toda ne smemo pozabiti, da operacijski sistem v realnem času postavlja omejitve, glede izvajanja opravil. Prekinitveni podprogram predstavlja eno izmed opravil, ki pa za razliko od navadnih opravil, lahko nastopa ob naključno določenih časih, kar lahko popolnoma podre časovno razporeditev ostalih opravil.

Obravnavana sta dva pristopa za uvedbo podpore prekinitvam in ob vsakem so podane njegove prednosti in slabosti. Kot osnovni kriteriji služijo naslednje karakteristike odziva na zahtevo po prekinitvi:

- hitrost odziva na zahtevo,
- število prekinitvenih zahtev, ki jih je sistem sposoben obdelati v enoti časa,
- vpliv posega v operacijski sistem na dolžino časovne rezine,
- poraba procesorskega časa za upravljanje s prekinitvami,
- komplikacije, ki nastanejo zaradi prekinitev.

Za izhodišče izvedbe posameznih pristopov je uporabljen obstoječi operacijski sistem učnih mikrokrmilniških sistemov. Določene funkcije, kot na primer maskiranje IRQ1 prekinitve, je potrebno izvesti zunaj mikrokrmilnika M6803.

Ključne besede:
operacijski sistemi v realnem času, mikrokrmilniki, časovno rezinjenje, prekinitve.

1 Abstract

The real-time operating system used by microcontroler systems RT68-01, RT68-02 and RT68-03 lacks the support for interrupt handling. Interrupts are used in almost every commercial microcontroler system. Therefore it has been decided to upgrade the existing real-time operating system so it would be capable of interrupt handling.

The M6803 microcontroler used in the above mentioned systems has the necessary hardware support. The nature of the existing real-time operating system imposes some serious limitations for interrupt handling. The interrupt handling subroutines are treated like all other processes. However there is one major difference: interrupt requests can occur at random times and are completely unpredictable. The latter fact makes it difficult to incorporate interrupt handling without loosing some of the advantages of the existing real-time operating system.

Two approaches are discussed. The major advantages and disadvantages of each approach are examined. The basic criteria used are:

- delay of the response to an interrupt request,
- number of interrupt requests per time unit the operating system can handle,
- impact on the length of the timeslice,
- percent of the time used for interrupt handling and
- complications arising from the introduction of interrupt handling to the existing operating system.

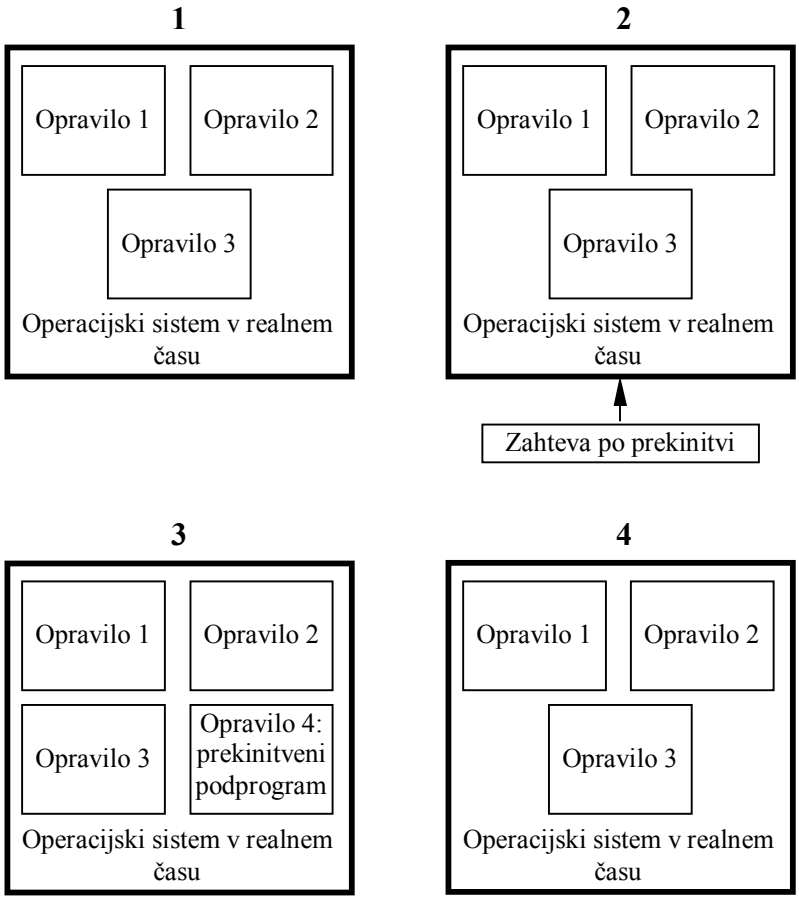
Certain operations (e.g. separately disabling IRQ1) are not hardware supported by the M6803 microcontroler and must be implemented by additional hardware.

Keywords:
real-time operating systems, microcontrolers, timeslicing, interrupts.

2 Uvod

Prekinitve [1] so pomemben sestavni del vsakega računalniškega sistema. V operacijskih sistemih v realnem času je obravnavanje zahtev po prekinitvi še posebej kritično, saj v veliki meri vpliva na robustnost sistema in hitrost odziva na signale iz okolice. Prekinitve so lahko šibka točka sistema, kot ilustrira naslednji zgled:

> Tik pred pristankom Apolla 11 na Luni je programska oprema računalnika sprožila alarm zaradi preobremenjenosti z opravili, kar je povzročilo, da se je računalnik vedno znova samodejno resetiral, da je lahko nemoteno izvajal najbolj kritična opravila. Ker je program kljub napakam še zmeraj deloval, so nadaljevali s pristankom. Pozneje so ugotovili, da so enemu od astronavtov pomotoma ukazali naj vklopi nek senzor, ki je neprestano pošiljal zahteve po prekinitvi procesorju in s tem povzročil preobremenitev sistema.
>
> Povzeto po [2].



Slika 1: Dogajanje v operacijskem sistemu v realnem času (opazovano s stališča okolice), ko ta prejme zahtevo po prekinitvi. 1: Operacijski sistem izvaja opravila v urniku. 2: Pojavi se zahteva po prekinitvi. 3: Operacijski sistem začne izvajati prekinitveni podprogram. 4: Prekinitveni podprogram zaključi z delom.

Procesor lahko naenkrat izvaja le eno opravilo. Navidezno hkratno izvajanje več opravil z enim samim procesorjem dosežemo tako, da procesor vsakemu opravilu posveti del svojega časa. Govorimo o kvazi-paralelizmu [2]. Če se opravila menjavajo dovolj hitro, dobimo vtis, da se izvajajo hkrati.

Vsak procesorski sistem komunicira s svojo okolico (zunanjimi enotami). Pri tem obstajata v splošnem dva možna pristopa:

- procesor sam ugotavlja, kdaj se mora posvetiti zunanji enoti s stalnim preverjanjem stanja zunanje enote (angl. *polling*),
- zunanja enota sporoča procesorju, kdaj naj se ukvarja z njo, tako da mu pošlje zahtevo po prekinitvi (angl. *interrupt*), nakar procesor prekine trenutno opravilo in se posveti zunanji enoti.

Pri sistemih v realnem času morajo biti rezultati opravil na voljo najkasneje v nekem vnaprej določenem času. Ker je število operacij, ki jih procesor lahko izvede v enoti časa omejeno, se zahtevam po prekinitvi ne da zmeraj ugoditi takoj in poljubno pogosto. Kdaj in kako pogosto so prekinitveni podprogrami izvedeni, določa v tem primeru kompromis med več zahtevami: koliko časa je preostalo procesorju od navadnih opravil, koliko časa potrebuje prekinitveni podprogram, da se izvede do konca in nenazadnje tudi od pomembnosti opravila in hitrega odziva na prekinitveno zahtevo (prioritete opravil).

Učni mikrokrmilniški sistemi RT68-01 [3], RT68-02 [4] in RT68-03 [5] so namenjeni spoznavanju temeljnih pristopov k realizaciji sistemov, ki delujejo v realnem času. Zasnovani so okoli danes že zastarelega mikrokrmilnika M6803, ki je bil izbran za jedro sistema zaradi enostavnosti. M6803 ima vse, kar je potrebno za realizacijo sistemov v realnem času, brez da bi ponujal različne kompleksne podsklope, ki sicer gradnjo mikrokrmilniških sistemov močno poenostavijo, zato pa zameglijo njihovo bistvo.

Jedro operacijskega sistema v realnem času [6] učnih mikrokrmilniških sistemov RT68-xx je zgrajeno na osnovi nekaterih predpostavk, ki poenostavljajo razumevanje njegovega delovanja. Ena temeljnih poenostavitev je, da ne dovolimo nobenih prekinitev razen prekinitve prostotekočega števca, ki služi deljenju časa na časovne rezine.

V nadaljevanju bomo poskušali to poenostavitev odpraviti in proučiti pristope za uvedbo podpore prekinitvam v operacijskem sistemu RT68. Kot glavni vodili nam bosta služili naslednji zahtevi:

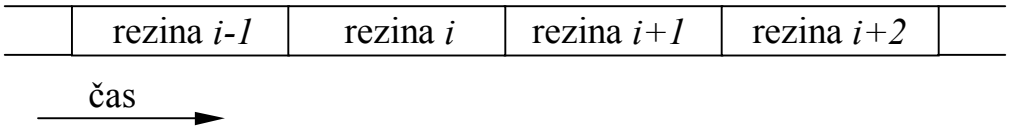
- sistem mora ostati čimbolj enostaven za razumevanje,
- podpora prekinitvam ne sme bistveno spremeniti zasnove operacijskega sistema RT68.

3 Operacijski sistem RT68

3.1 Izvedba večopravilnosti v operacijskem sistemu RT68

Večopravilnost omogoča nadzorni program [7], ki razporeja procesorski čas med opravila (razvrščevalnik, angl. *scheduler*). Opravila so razporejena v urniku, ki se izvaja krožno. Procesorski čas je razdeljen na časovne rezine (angl. *timeslice*). Vsakemu opravilu v urniku pripada pri enem obhodu urnika ena časovna rezina. Razvrščevalnik je zgrajen na osnovi štirih predpostavk:

1. vse časovne rezine so natanko enako velike (*1200* strojnih ciklov), zaradi česar je delitev procesorskega časa enostavno izvesti z uporabo prostotekočega števca,
2. vsako opravilo se zaključi še pred iztekom časovne rezine, kar zagotavlja, da dve opravili ne moreta hkrati dostopati do iste zunanje enote, poleg tega pa nam prihrani zamudno shranjevanje stanja opravila, ki je bilo prekinjeno zaradi izteka časovne rezine,
3. urnik opravil obsega *16* opravil, ki se izvajajo ciklično ter
4. zunanje enote ne povzročajo prekinitev.

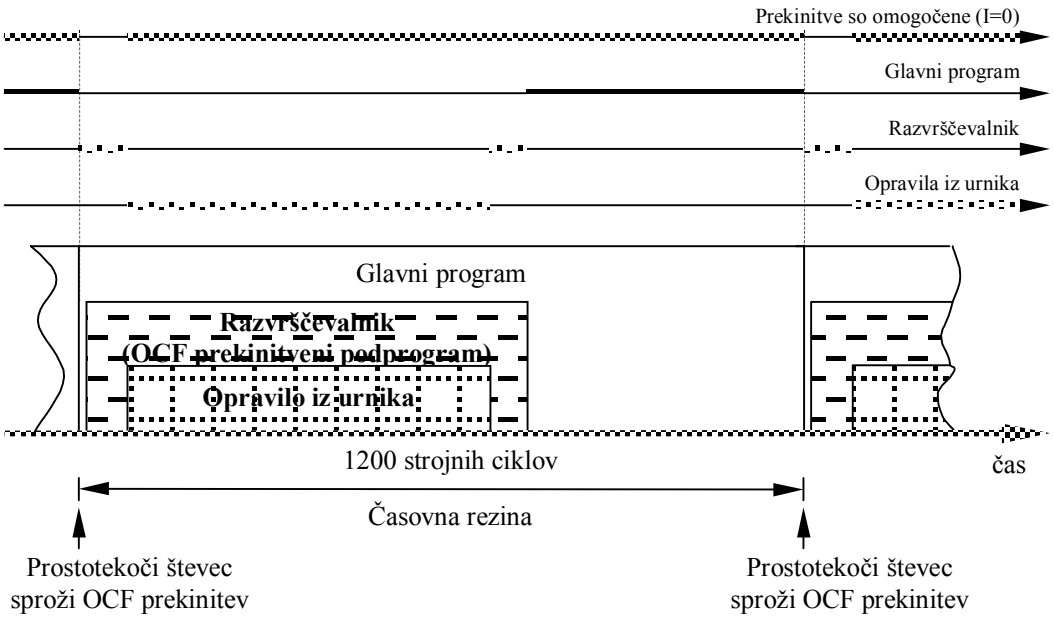


Slika 2: Delitev procesorskega časa na časovne rezine.

Časovna rezina se začne, ko prostotekoči števec na vsakih *1200* strojnih ciklov sproži OCF (*Output Compare Flag*) prekinitev. Ob skoku v prekinitveni podprogram se samodejno onemogočijo prekinitve. Prekinitveni podprogram je razvrščevalnik, ki najprej preveri, če se je opravilo iz prejšnje časovne rezine zaključilo. Če se ni, je bila prekršena druga predpostavka in zato prekine izvajanje programa ter sporoči napako. Če temu ni tako, nastavi čas za naslednjo OCF prekinitev, omogoči prekinitve in skoči v opravilo, ki je na vrsti. Ko se opravilo konča, se izvajanje vrne v razvrščevalnik, ki nato vrne nadzor glavnemu programu do naslednje OCF prekinitve.

Ob zagonu procesorja se začne izvajati glavni program, ki najprej vzpostavi začetno stanje v spremenljivkah razvrščevalnika in ga nato požene. Od tega trenutka naprej se glavni program prekine na vsakih *1200* strojnih ciklov. Ob prekinitvi procesor začne izvajati razvrščevalnik, ki požene tekoče opravilo. Preostali procesorski čas v časovni rezini, ki ga ne porabita razvrščevalnik in tekoče opravilo, je na voljo glavnemu programu. Ker se ob vsakem skoku v razvrščevalnik samodejno shrani stanje procesorja na sklad in ob vrnitvi iz razvrščevalnika prebere s sklada, izvajanje opravil v urniku nima neposrednega vpliva na tok glavnega programa. Ta se izvaja v vrzelih med posameznimi opravili.

Razvrščevalnik porabi približno *8%* procesorskega časa v časovni rezini.



Slika 3: Zgradba časovne rezine pri izhodiščni različici operacijskega sistema RT68.

3.2 Splošno o prekinitvah in sistemih v realnem času

Zahteva po prekinitvi lahko nastopi kadarkoli. Nanjo se procesor lahko odzove, ko je dotična prekinitev omogočena in globalna prekinitvena maska ni postavljena. Če eden od teh dveh pogojev ni izpolnjen, se procesor na zahtevo odzove takoj, ko sta oba pogoja spet izpolnjena. V primeru, da se pojavi več zahtev po prekinitvi naenkrat, najprej ustreže tisti, ki ima višjo prioriteto.

Procesor se odzove tako, da shrani svoje stanje na sklad, onemogoči nekatere ali pa celo vse prekinitve in skoči v ustrezni prekinitveni podprogram. Da je prekinitve sploh smiselno uporabljati, moramo operacijski sistem zasnovati tako, da se prekinitveni podprogram izvrši takoj, ali pa vsaj čimprej po nastopu zahteve. Ko se prekinitveni podprogram zaključi, procesor prebere stanje s sklada in nadaljuje z delom. Če znotraj prekinitvenih podprogramov dovolimo prekinitve, govorimo o gnezdenju prekinitev. Tak primer imamo v operacijskem sistemu RT68, saj so prekinitve med izvajanjem tekočega opravila, ki se izvaja znotraj OCF prekinitvenega podprograma, omogočene.

Obravnavali bomo dva pristopa za uvedbo prekinitev v operacijski sistem RT68:

1. Časovne rezine imajo vse enako dolžino (*1200* strojnih ciklov). V vsaki časovni rezini je del časa rezerviran za prekinitvene podprograme. Govorimo o ***statičnem dodeljevanju procesorskega časa***.
2. Časovne rezine so lahko daljše ali krajše od *1200* strojnih ciklov. Zahtevamo le, da povprečna frekvenca izvajanja opravil ostane dolgoročno enaka. Zato nekatere časovne rezine skrajšamo, da kompenziramo podaljšanje prejšnjih časovnih rezin zaradi izvajanja prekinitvenih podprogramov. Pravimo tudi, da je procesorski čas ***dinamično dodeljen***.

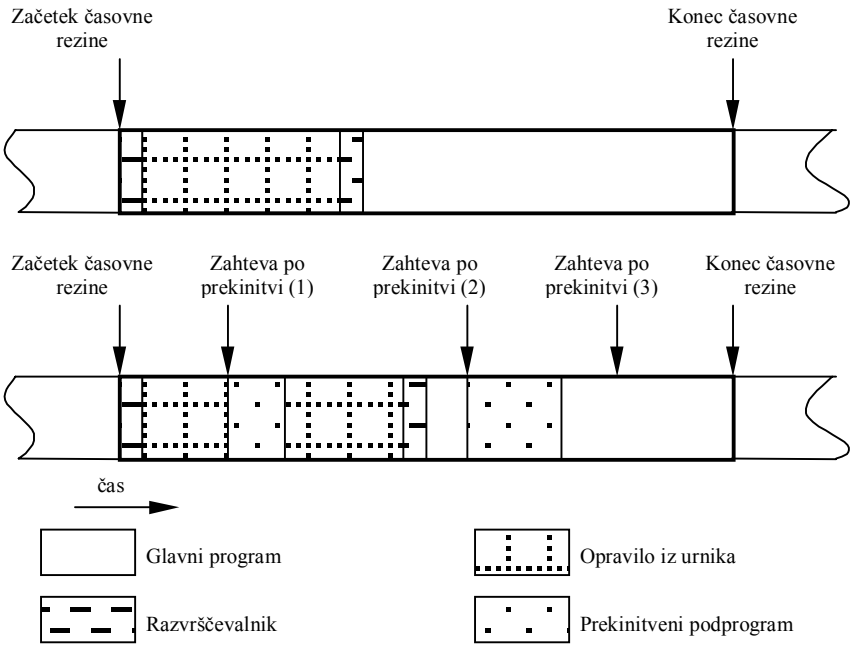
3.3 Statično dodeljevanje procesorskega časa

Vsa opravila morajo biti krajša od

$$O = 1200 - R - P \qquad (1)$$

O cikli, ki so na voljo za opravilo,
R cikli, ki jih porabi razvrščevalnik,
P cikli, ki so na voljo prekinitvenim podprogramom,

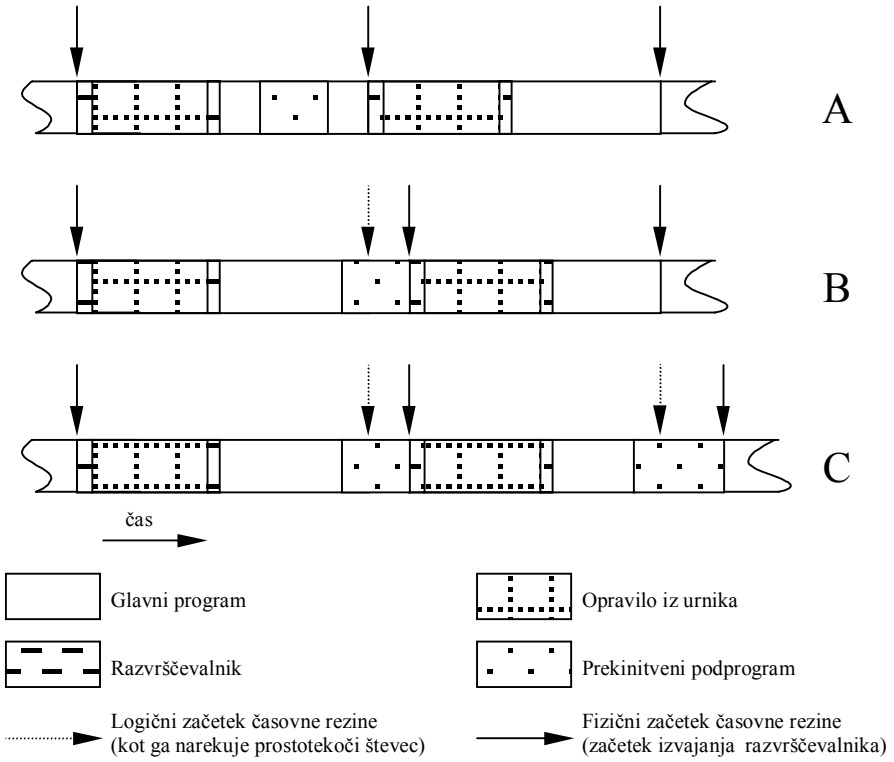
ker ne vemo, v kateri časovni rezini bo nastopila zahteva po prekinitvi. V vsaki časovni rezini je rezervirano P strojnih ciklov za izvajanje prekinitvenih podprogramov. Če zahteva po prekinitvi ne nastopi, so ti cikli na razpolago glavnemu programu.



Slika 4: Statično dodeljevanje procesorskega časa.
Zgoraj: časovna rezina, v kateri ne nastopi zahteva po prekinitvi.
Spodaj: ista časovna rezina s tremi zahtevami po prekinitvi. Procesor se na prvi dve zahtevi odzove takoj. Ker sta prva dva prekinitvena podprograma rezervirala zase toliko časa, da ga ni ostalo dovolj za še en prekinitveni podprogram, se procesor odzove na tretjo zahtevo po prekinitvi šele v naslednji časovni rezini.

Če so prekinitve omogočene, globalna prekinitvena maska ni postavljena in nastopi zahteva po prekinitvi, se procesor takoj odzove na zahtevo s postavitvijo stanja na sklad in skokom v prekinitveni podprogram.

Na začetku vsakega prekinitvenega podprograma zmanjšamo število rezerviranih strojnih ciklov za toliko, koliko jih nameravamo porabiti. Če je preostanek premajhen, da bi se v isti časovni rezini lahko izvršil najdaljši možni prekinitveni podprogram, prekinitve onemogočimo. Ob začetku nove časovne rezine razvrščevalnik postavi število rezerviranih strojnih ciklov nazaj na začetno vrednost in omogoči prekinitve.

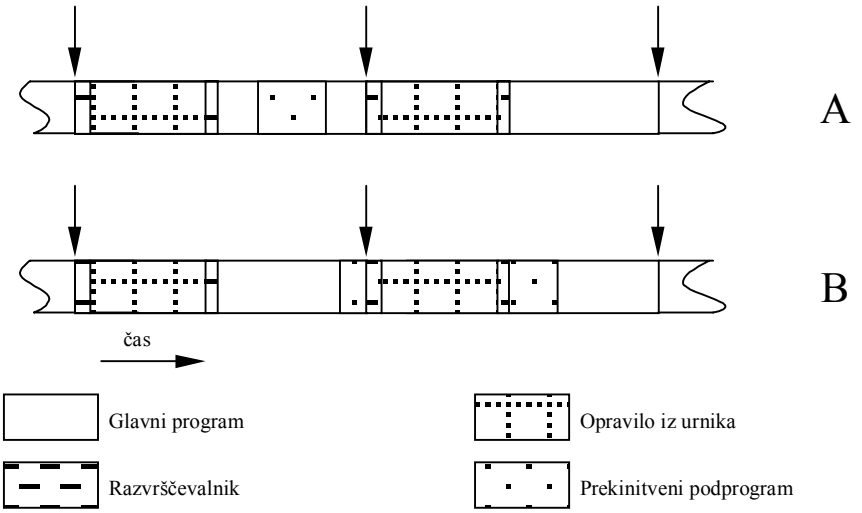


Slika 5: V primeru, da prekinitveni podprogram ne seže v naslednjo časovno rezino, se ta ne zakasni (A). Če nastopi zahteva po prekinitvi proti koncu časovne rezine, gnezdenje prekinitev pa ni dovoljeno, se naslednja časovna rezina začne z zakasnitvijo (B). Če v naslednji časovni rezini prekinitveni podprogram ne preseže logičnega konca rezine, se stanje normalizira (B). Začetki časovnih rezin so zakasnjeni, dokler prekinitveni podprogrami presegajo logične konce časovnih rezin (C).

Slika 5 opozarja na posebnost opisanega pristopa. Če se zahteva po prekinitvi pojavi malo pred koncem časovne rezine, rezervirani strojni cikli pa še niso porabljeni, se prekinitveni podprogram izvede in seže v naslednjo časovno rezino. Če ne dovolimo gnezdenja prekinitev, se razvrščevalnik začne izvajati z zakasnitvijo. Začetek izvajanja razvrščevalnika ostaja zakasnjen v primeru, da se situacija ponovi v naslednji časovni rezini. Če temu ni tako, je naslednja časovna rezina krajša in stanje se normalizira na račun skrajšanja časovne rezine. Začetek časovne rezine je lahko zakasnjen za največ *P* strojnih ciklov.

Povedano velja le za primer, ko gnezdenje prekinitev ni dovoljeno. Če je gnezdenje prekinitev dovoljeno, se izvajanje prekinitvenega podprograma prekine z OCF prekinitvijo in razvrščevalnik s tekočim opravilom se izvede znotraj prekinitvenega podprograma. Šele ko se izvajanje razvrščevalnika zaključi z ukazom `RTI`, se nadaljuje izvajanje prekinitvenega podprograma. Začetek časovne rezine ni zakasnjen. Pri gnezdenju prekinitev moramo biti previdni, saj ga smemo dovoliti šele takrat, ko prekinitveni podprogram odstrani zahtevo po prekinitvi. Če se tega ne držimo, lahko en in ista zahteva po prekinitvi še enkrat sproži prekinitveni podprogram ob začetku izvajanja opravila v razvrščevalniku.

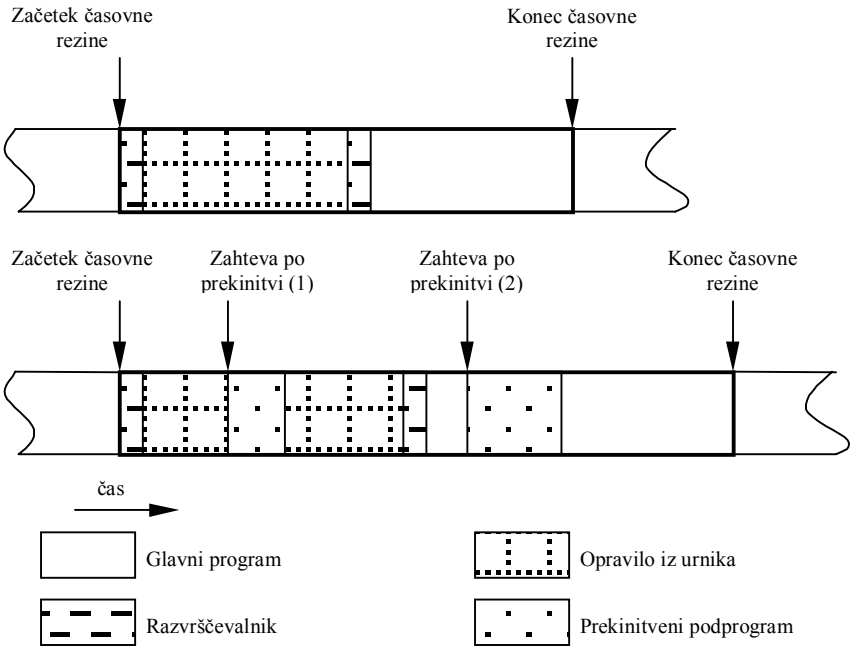
Če gledamo na sistem s stališča zunanjega opazovalca, imajo prekinitveni podprogrami prednost pred opravili iz urnika, če gnezdenje prekinitev ni dovoljeno.



Slika 6: Če je gnezdenje prekinitev dovoljeno in se zahteva po prekinitvi pojavi proti koncu časovne rezine, se pripadajoči prekinitveni podprogram prekine z OCF prekinitvijo, opravilo v naslednji časovni rezini pa se izvede nezakasnjeno (B). Prekinitveni podprogram se nadaljuje, ko razvrščevalnik zaključi z delom.

3.4 Dinamično dodeljevanje procesorskega časa

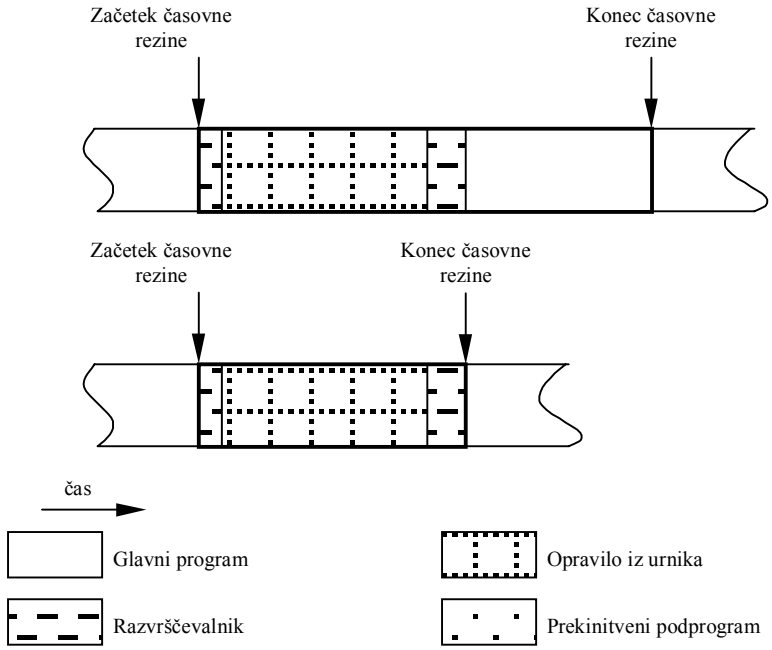
Druga možnost je bolj zapletena. Njena pomanjkljivost je v tem, da za posamezne rezine ne moremo natančno določiti njihovih začetkov. Ko nastopi zahteva po prekinitvi, tekočo časovno rezino podaljšamo za toliko strojnih ciklov, koliko jih prekinitveni podprogram potrebuje. To ima za posledico zaostanek pri izvajanju opravil, ki ga nadoknadimo s krajšanjem časovnih rezin, ki sledijo. Rezin ne moremo podaljševati v nedogled, saj bi se s tem dolgoročno zmanjšala frekvenca izvajanja opravil v urniku. Ko zaostanek preseže *M*, prekinitve onemogočimo. Prekinitve ostajajo onemogočene, dokler zaostanek ne pade pod *m*, ko jih spet omogočimo.



Slika 7: Dinamično dodeljevanje procesorskega časa.
Zgoraj: časovna rezina, v kateri ne nastopi zahteva po prekinitvi.
Spodaj: ista časovna rezina, v kateri sta nastopili dve zahtevi po prekinitvi. Rezina je bila podaljšana za toliko strojnih ciklov, koliko jih porabita oba prekinitvena podprograma.

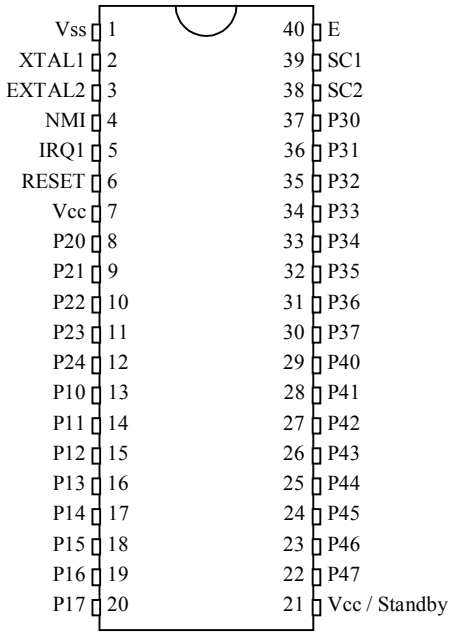
Že na prvi pogled lahko ugotovimo, da tak pristop zahteva dosti večje »režijske stroške«. Čas, ki ga porabi operacijski sistem zase, je daljši, zato pa je sistem kratkoročno sposoben obdelati veliko število zahtev po prekinitvi.

Tako statično, kot dinamično dodeljevanje procesorskega časa zahtevata, da programer za vsak prekinitveni podprogram navede, koliko strojnih ciklov bo ta rezerviral zase. Gotovo bi bilo dosti bolj elegantno ugotoviti stanje prostotekočega števca na začetku in koncu prekinitvenega podprograma, nakar bi lahko iz razlike obeh vrednosti sklepali na število porabljenih strojnih ciklov. Tak pristop zahteva precej več operacij. Kot bomo videli pozneje, je že samo dinamično dodeljevanje procesorskega časa precej požrešno opravilo. Ker smo zaradi počasnosti uporabljenega mikrokrmilnika v sistemih RT68-xx precej omejeni, bi samodejno ugotavljanje števila porabljenih strojnih ciklov pustilo bistveno manj časa za opravila v urniku.



Slika 8: Krajšanje časovne rezine.
Zgoraj: neskrajšana časovna rezina.
Spodaj: »Izposojeni« čas v rezinah, ki so jih prekinile zahteve po prekinitvi, nadoknadimo s krajšanjem časovnih rezin, ki sledijo. Pri tem odvzamemo procesorski čas glavnemu programu.

4 Nekaj podrobnosti o mikrokrmilniku M6803



Slika 9: Priključki M6803.

4.1 Zahteve po prekinitvi

Ko M6803 dobi katero od zahtev po prekinitvi, shrani stanje procesorja (registre PC, X, A, B, in CC) na sklad, onemogoči prekinitve s postavitvijo globalne prekinitvene maske in skoči v prekinitveni podprogram, katerega naslov prebere iz pripadajočega prekinitvenega vektorja. Če naenkrat nastopi več različnih zahtev po prekinitvi, se najprej upošteva zahteva z najvišjo prioriteto.

M6803 lahko ignorira zahteve po prekinitvah, če z ukazom `SEI` postavimo bit I v registru stanja (CC) na vrednost '1'. Pravimo, da smo maskirali prekinitve. Bit I je globalna prekinitvena maska. Prekinitve spet omogočimo z ukazom `CLI`. Povedano ne velja za prekinitvi RESET in NMI, ki ju ne moremo maskirati. Vse prekinitve, ki jih lahko maskiramo z globalno prekinitveno masko, lahko maskiramo tudi vsako posebej s postavitvijo ustreznih bitov v kontrolnih registrih procesorja. Izjema je le IRQ1, ki jo lahko posebej maskiramo le z ustreznim zunanjim vezjem. Več o maskiranju prekinitev je napisanega v razdelku 4.2.

Prekinitve so lahko prožene nivojsko (*level*) ali robno (*edge*).

Zahteva po nivojsko proženi prekinitvi nastopi, ko signal, ki proži prekinitev doseže aktivni nivo (za nekatere je to '0', za nekatere pa '1'). Zahteve po prekinitvi prihajajo, dokler je aktivni nivo prisoten.

Zahteva po robno proženi prekinitvi nastopi ob prehodu pripadajočega signala iz stanja '0' v stanje '1' ali pa ob prehodu iz stanja '1' v stanje '0'. Prehod, ki sproži prekinitev, lahko v nekaterih primerih izberemo programsko, v drugih pa je določen z zgradbo procesorja.

M6803 premore naslednje prekinitve (urejene po prioriteti od najvišje proti najnižji):

- RESET- izvede se, ko napetost na nožici *6* preseže *4V*. Vektor kaže na začetek programa, ki ga izvaja procesor in je zapisan na naslovu `$FFFE`.
- NMI (*Non Maskable Interrupt*) - robno prožena prekinitev, do katere pride ob prehodu z visokega na nizek nivo na nožici *4*. Vektor na prekinitveni podprogram je na naslovu `$FFFC`.
- SWI (*SoftWare Interrupt*) - do te prekinitve pride, ko procesor naleti na ukaz `SWI`. Vektor na prekinitveni podprogram je na naslovu `$FFFA`.
- IRQ1 (*Interrupt ReQuest 1*) - je nivojsko prožena prekinitev. Sproži jo nizek nivo na nožici *5*. Vektor na prekinitveni podprogram je na naslovu `$FFF8`.
- IS3 - je robno prožena prekinitev. Dostopna je le v »single-chip« načinu delovanja. Čeprav M6803 v sistemih RT68-xx ne deluje v tem načinu bomo povedali nekaj tudi o tej prekinitvi. Prehod iz '1' v '0' na vhodu IS3 (nožica *39*) povzroči, da se bit *7* v nadzorno-statusnem registru vrat *3* (P3CSR) postavi na '1', kar sproži prekinitev IRQ1. Prekinitev IS3 je torej dejansko le drugače (robno) prožena prekinitev IRQ1.
- IRQ2 (*Interrupt ReQuest 2*) - je interna prekinitvena linija, ki ni dostopna od zunaj. Prekinitev je nivojsko prožena. Nanjo so vezani štirje viri prekinitev, ki so prav tako urejeni po prioriteti:
  - ICF (*Input Capture Flag*) - prekinitev ima vektor na naslovu `$FFF6`. Sproži se, ko se ob ustreznem prehodu (bodisi '0' v '1' ali '1' v '0') na nožici *8* (P20) postavi bit *7* (ICF) registra TCSR na naslovu `$08` na '1'. Tedaj se stanje *16*-bitnega prostotekočega števca na naslovu `$09` prenese v register ICR na naslovu `$0D`. Stanje P20 se preverja ne glede na to ali je P20 konfiguriran kot vhod ali izhod.
  - OCF (*Output Compare Flag*) - prekinitev se sproži, ko prostotekoči števec doseže vrednost zapisano na naslovu `$0B`. Tedaj se bit *6* (OCF) registra TCSR postavi na '1', kar sproži OCF prekinitev. Vektor na prekinitveni podprogram se nahaja na naslovu `$FFF4`. To prekinitev uporablja operacijski sistem RT68 za deljenje časa na časovne rezine.
  - TOF (*Timer Overflow Flag*) - prekinitev se sproži, ko prostotekoči števec doseže vrednost `$FFFF`, kar postavi bit *5* (TOF) registra TCSR na '1' in sproži TOF prekinitev. Vektor na prekinitveni podprogram se nahaja na naslovu `$FFF2`.
  - SCI (*Serial Communication Interface*) – M6803 ima vgrajeni serijski komunikacijski vmesnik, ki je prav tako lahko vir zahtev po prekinitvi. Prekinitev se lahko sproži ob polnem sprejemnem registru, praznem oddajnem registru, napaki prekrivanja ali napaki okvirja. Vektor na prekinitveni podprogram se nahaja na naslovu `$FFF0`.

4.2 Maskiranje prekinitev

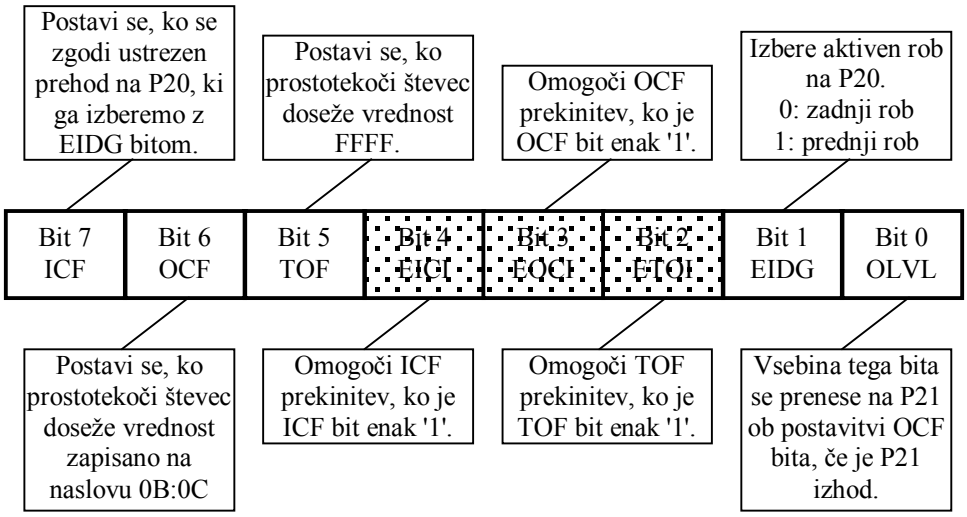
Prekinitve lahko onemogočimo na dveh ravneh:

- Globalno s postavitvijo globalne prekinitvene maske (ukaz `SEI`), kar onemogoči vse prekinitve razen RESET in NMI. V operacijskem sistemu RT68 tega ne smemo kadarkoli narediti, ker s tem prekinemo tudi časovno rezinjenje, ki ga proži OCF prekinitev.
- Posamično s postavljanjem ustreznih bitov v nadzornih registrih procesorja, kar je mogoče za vse prekinitve, razen za RESET, NMI, SWI in IRQ1.

4.2.1 Prostotekoči števec in maskiranje *ICF*, *OCF* in *TOF* prekinitev:

Prostotekoči števec (FRC) je *16*-bitni register na naslovu `$09`, katerega vsebina se poveča za ena ob vsakem strojnem ciklu. Primerjalni register (OCR) prostotekočega števca je prav tako *16*-biten register, ki se nahaja na naslovu `$0B`.

Ko je vsebina prostotekočega števca enaka vsebini primerjalnega registra, se postavi OCF bit v registru TCSR na '1'. Stanje OCF bita se lahko uporablja za proženje OCF prekinitve. OCF prekinitev onemogočimo s postavitvijo EICI bita v registru TCSR na '0'. V operacijskem sistemu RT68 je ta prekinitev uporabljena za delitev časa na časovne rezine.



Slika 10: Nadzorno statusni register prostotekočega števca (TCSR) na naslovu `$08`.

Ko prostotekoči števec doseže vrednost `$FFFF`, se postavi TOF bit registra TCSR na '1'. Stanje TOF bita se lahko uporablja za proženje TOF prekinitve. TOF prekinitev onemogočimo s postavitvijo bita ETOI v registru TCSR na '0'.

Z bitom EIDG izbiramo aktiven prehod na vratih P20 ob katerem se postavi ICF bit na '1' in vsebina prostotekočega števca zapiše v register ICR na naslovu `$0D`. Če so vrata P21 definirana kot izhod, se vsebina OLVL bita registra TCSR prenese na P21. Stanje ICF bita lahko proži ICF prekinitev, ki jo lahko onemogočimo s postavitvijo EICI bita v registru TCSR na '0'.

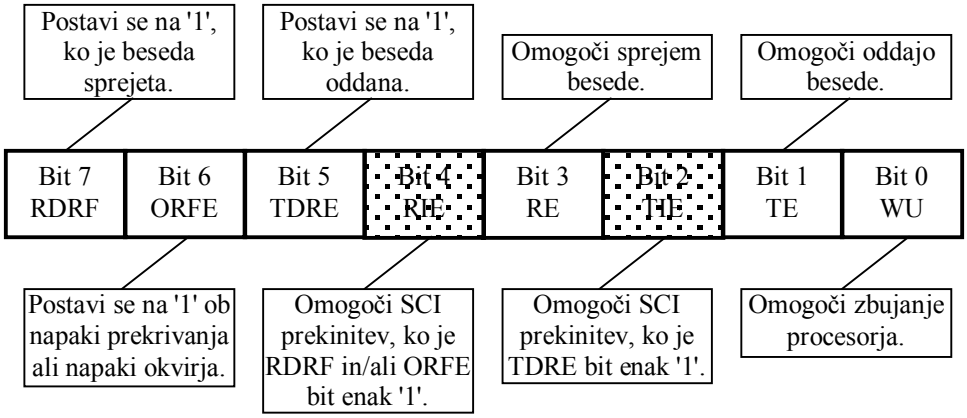
Biti ICF, OCF in TOF prožijo pripadajoče prekinitve (če so le te omogočene), dokler so enaki '1'. Vsi trije se postavijo na '0' ob resetu ali pa:

- ICF ob branju TCSR in nato branju zgornjega byta ICR (na naslovu `$0D`),
- OCF ob branju TCSR in nato pisanju v OCR (na naslovu `$0B` ali `$0C`) in
- TOF ob branju TCSR in branju zgornjega byta FRC (na naslovu `$09`).

4.2.2 *SCI* prekinitev

M6803 ima vgrajen asinhroni serijski vmesnik, ki je lahko vir SCI prekinitev. SCI prekinitev se lahko zgodi, ko:

- je v sprejemni register prenešena pravkar sprejeta beseda,
- se prične oddaja besede v oddajnem registru,
- pride do napake prekrivanja ali
- napake okvirja.



Slika 11: Sprejemno/oddajni nadzorno statusni register (TRCSR) na naslovu `$11`.

Ko je sprejeta cela beseda, se RDRF bit postavi na '1'. RDRF bit lahko proži SCI prekinitev.

Če pri sprejemu pride do napake prekrivanja ali pa do napake okvirja, se ORFE bit postavi na '1'. Tudi ORFE bit lahko proži SCI prekinitev.

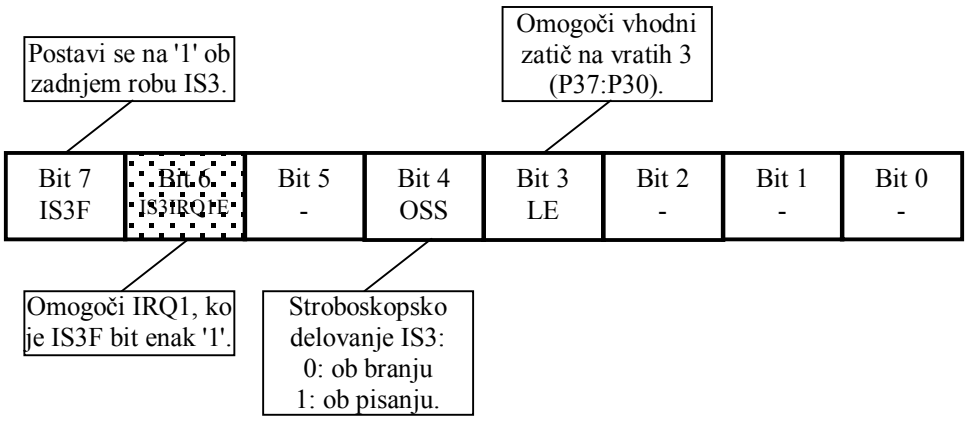
Proženje SCI prekinitve z RDRF in/ali ORFE bitom lahko maskiramo s postavitvijo RIE bita na '0'.

Po končani oddaji besede se TDRE bit postavi na '1'. Z njim lahko prožimo SCI prekinitev. Proženje prekinitve preprečimo s postavitvijo bita TIE na '0'.

Biti RDRF, ORFE in TDRE prožijo SCI prekinitev (seveda samo če je to dovoljeno z RIE in TIE bitoma) dokler so enaki '1'. Ob resetu se vsi trije postavijo na '0'. Brišemo jih lahko tudi posamično:

- RDRF z branjem TRCSR in nato branjem sprejete besede na naslovu `$12`,
- ORFE z branjem TRCSR in nato branjem sprejete besede na naslovu `$12` in
- TDRE z branjem TRCSR in nato vpisom besede za oddajo na naslov `$13`.

4.2.3 *IS3* prekinitev

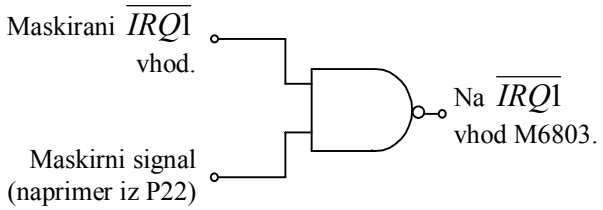


Slika 12:Nadzorno statusni register vrat *3* (P3CSR) na naslovu `$0F`.

Ker deluje M6803 v RT68-xx sistemih v multipleksiranem načinu, so vrata *3* uporabljena za podatkovno in spodnji del naslovnega vodila. IS3 vhod, ki je v »single-chip« načinu dostopen na nožici *39*, v tem načinu delovanja ni dostopen. Slika 12 pove praktično vse o proženju prekinitve IRQ1 z IS3 vhodom. Omeniti velja, da IS3F bit brišemo z branjem P3CSR in nato branjem ali pisanjem podatkovnega registra vrat *3* na naslovu `$06`.

4.2.4 *IRQ1* prekinitev

IRQ1 prekinitve ne moremo programsko maskirati drugače, kot s postavitvijo globalne prekinitvene maske, kar pa maskira tudi ostale prekinitve. Ker tega ne smemo storiti, saj bi ustavili deljenje procesorskega časa na časovne rezine, moramo uporabiti preprosto dodatno vezje:



Slika 13: Maskiranje IRQ1 prekinitve z dodatnim vezjem.

4.2.5 *SWI* prekinitev

SWI prekinitev je bolj prekinitev »po svojem imenu«, kot prava prekinitev. Prožimo jo namreč z ukazom SWI, ki je v bistvu enak ukazu `JSR`, le da postavi celotno stanje procesorja na sklad in skoči na naslov zapisan v prekinitvenem vektorju `$FFFA`. Iz prekinitvenega podprograma se vračamo z ukazom `RTI`, kot iz vseh drugih prekinitvenih podprogramov. Pri tej prekinitvi tudi nima smisla govoriti o maskiranju, saj programer natančno ve, kdaj bo skočil v SWI prekinitveni podprogram in lahko SWI prekinitev obravnava na enak način kot skok v podprogram.

4.2.6 Prekinitvi *RESET* in *NMI*

Nobene od teh dveh ne moremo maskirati. Prekinitev RESET je zmeraj uporabljena za zagon procesorja. Prekinitev NMI običajno uporabljamo za najpomembnejša sporočila procesorju, ki nastopajo le v izjemnih okoliščinah. Primer uporabe NMI prekinitve je sporočanje procesorju, da napajalna napetost upada. V NMI prekinitveni podprogram zapišemo ukaze, ki naj jih procesor izvede ob izkopu napajanja.

5 Operacijski sistem RT68 s podporo prekinitvam

V tem razdelku bomo navedli tiste dele, ki jih je treba spremeniti oziroma dodati izhodiščni različici operacijskega sistema RT68. Obe (izhodiščna in nadgrajena) različici se v celoti nahajata na priloženi disketi.

5.1 Odčitavanje prostotekočega števca

Prvi problem, na katerega velja opozoriti, je odčitavanje prostotekočega števca (FRC). Potrebujemo ga pri preprečevanju hkratnega dostopa do skupnih sredstev in krajšanju časovnih rezin. V dosedanjih različicah operacijskega sistema RT68 je napaka, ki sicer redko pride do izraza in jo je zaradi tega tudi toliko težje odkriti. Njen izvor je v načinu branja *16*-bitnega FRC registra z ukazom `LDD`, ki traja *4* strojne cikle:

1. branje *8*-bitne kode operacije,
2. branje *8*-bitnega naslova *M,*
3. prepis vsebine na naslovu *M* v višji del registra D,
4. prepis vsebine na naslovu *M+1* v nižji del registra D.

Med tretjim in četrtim korakom se stanje FRC poveča za ena. V primeru, da imamo prenos iz nižjih *8* bitov (primer `$XXFF+1⇒$YY00`), je odčitek premajhen za `$FF`. Če pa nastopi prenos tudi iz višjih *8* bitov (primer `$FFFF+1⇒$0000`), je odčitek premajhen za kar `$FFFF`.

Postopek za zanesljivo branje FRC je naslednji:

```
        ldd   _FRC        ; 4
        tstb              ; 2
        bne   SYNOK       ; 3
        ldd   _FRC        ; 4  D <- FRC v 4. ciklu ukaza
        bra   SYNCORR     ; 3
SYNOK   addd  #$0009      ; 4  Popravljanje odčitka in
        bra   SYNCORR     ; 3  zakasnitev dolžine 7 ciklov
SYNCORR ...
```

Najprej preberemo vsebino FRC. Če je nižji byte enak `$00`, branje ponovimo. Če je bilo prvo branje uspešno, porabimo manj strojnih ciklov. Tudi odčitana vrednost FRC se nanaša na drug trenutek. Zaradi tega je potrebno odčitek popraviti (prišteti *9*) in ob tem porabiti *7* strojnih ciklov (kar je tudi razlog za navidez nesmiseln ukaz `bra SYNCORR`, ki sledi popravljanju odčitka). Tako dobljeni odčitek je enak za *4* zmanjšani vsebini FRC v trenutku, ko teče prvi strojni cikel ukaza na naslovu `SYNCORR`. Branje FRC traja v vsakem primeru *16* strojnih ciklov. Na koncu postopka je odčitek shranjen v registru D.

5.2 Problem hkratnega dostopa do skupnih sredstev

Ko bomo govorili o sredstvih, bomo v prvi vrsti mislili na vsebino skupine pomnilnih celic, do katere tekom izvajanja programa dostopa več opravil. Seveda do hkratnega dostopa v dobesednem pomenu ne more priti, saj v našem sistemu ne vlada popoln paralelizem (več procesorjev). Govorimo o kvazi-paralelizmu, ki smo ga realizirali s časovnim rezinjenjem. Ker pa se v našem sistemu dogajajo prekinitve, lahko pride do situacije, ki jo procesi »čutijo« kot hkraten dostop.

Na primer: del programa popravlja čas zapisan na štirih naslovih (ure, minute, sekunde in *1/256* sekunde). Ko je vsebina ur in minut že popravljena, pride do zahteve po prekinitvi in

prekinitveni podprogram prebere čas. Za ure in minute bo prebral novo vrednost, za sekunde in *1/256* sekunde pa staro. Problem je podoben, kot pri branju vsebine prostotekočega števca.

Dve opravili iz urnika ne moreta hkrati dostopati do istega sredstva, saj se vsako opravilo izteče še pred koncem časovne rezine. Primeri pa se lahko:

1. opravilo iz urnika in glavni program hkrati dostopata do istega sredstva,
2. prekinitveni podprogram in glavni program hkrati dostopata do istega sredstva,
3. prekinitveni podprogram in opravilo iz urnika hkrati dostopata do istega sredstva,
4. dva prekinitvena podprograma hkrati dostopata do istega sredstva (v primeru, da dovolimo gnezdenje prekinitev).

Dokler v operacijski sistem ni bila vgrajene podpora prekinitvam, se je lahko zgodil le prvi primer. Težave so se dale razrešiti s klicanjem podprograma, ki je vrnil nadzor glavnemu programu šele potem, ko je bilo na voljo želeno število neprekinjenih strojnih ciklov, kar je podprogram ugotovil z branjem prostotekočega števca in primerjanjem le tega z vsebino registra OCR. Ker je v [7] uporabljen podprogram, ki ne prebere pravilno prostotekočega števca, je popravljena različica podana v dodatku.

Z uvedbo prekinitev pa se lahko izvajanje glavnega programa in opravil iz urnika kadarkoli prekine in obstaja nevarnost hkratnega dostopa. Prej uporabljeni postopek odpove.

Obstaja rešitev, ki je dokaj kratka, enostavna in zato primerna za uporabo v sistemih RT68-xx. Preden želimo dostopati do skupnih sredstev, onemogočimo prekinitve. Po končanem branju ali pisanju prekinitve spet omogočimo. Kot primer nam bo služilo branje časa:

```
sei
ldaa  URE
ldab  MINUTE
cli
```

Pristop uporabljamo v glavnem programu, v opravilih iz urnika in prekinitvenih podprogramih v katerih je dovoljeno gnezdenje prekinitev. Enostavnost take rešitve pogojuje tudi njeno največjo hibo. Med branjem so prekinitve onemogočene, kar ima za posledico daljši odzivni čas na zahtevo po prekinitvi. Pri uporabi opisanega postopka se lahko zgodi, da pride do prekoračitve dolžine časovne rezine, če se pojavi zahteva po OCF prekinitvi medtem, ko so prekinitve onemogočene. Razvrščevalnik tega ne »opazi« in se začne izvajati šele, ko odstranimo globalno prekinitveno masko z ukazom `CLI`. Zaradi zakasnitve začetka izvajanja razvrščevalnika je naslednja časovna rezina krajša. Do težav lahko pride, če je opravilo v urniku daljše od

$$O_{\max} = O - M_{\max} \qquad (2)$$

O strojni cikli, ki so na voljo opravilu iz urnika,
M_{max} najdaljši dostop do skupnega sredstva v glavnem programu.

Težavam se izognemo tako, da vsa opravila napišemo krajša od O_{max} strojnih ciklov.

V večjih operacijskih sistemih v realnem času [8] se za preprečevanje hkratnega dostopa uporabljata strukturi semafor in monitor, ki sta opisani v [2]. Ker smo v sistemih RT68-xx zelo omejeni s procesorskim časom, ju nismo realizirali.

5.3 Onemogočanje prekinitev

Naslednji podprogram onemogoči vse prekinitve in shrani stanje (prekinitev je/ni omogočena) v RAM pomnilniku:

```
                                ; 6 Ciklov za klic podprograma. DISINTS ldaa
_P3CSR        ; 3 Shrani stanje IS3-IRQ1 v
        tab                     ; 2 I_P3CSR in onemogoči IS3.
        anda #%01000000         ; 2
        andb #%10111111         ; 2
        staa I_P3CSR            ; 4
        stab _P3CSR             ; 3
        ldaa _TCSR              ;    Shrani stanje ICF in TOF v
        tab                     ;    I_TCSR in ju onemogoči.
        anda #%00010100         ;
        andb #%11101011         ;
        staa I_TCSR             ;
        stab _TCSR              ;
        ldaa TRCSR              ;    Shrani stanje SCI v I_TRCSR in
        tab                     ;    jo onemogoči.
        anda #%00010100         ;
        andb #%11101011         ;
        staa I_TRCSR            ;
        stab _TRCSR             ;
        ldaa P2DR               ;    Shrani stanje IRQ1 v I_P2DR in
        tab                     ;    jo onemogoči (Deluje samo, če
        anda #%00000100         ;    je dodano vezje za onemogočanje
        andb #%11111011         ;    IRQ1.).
        staa I_P2DR             ;
        stab _P2DR              ;
        ldaa #%00111111         ; 2 Prikaži 'O' na 7-segmentnem
        staa $A000              ; 4 prikazovalniku (sistem je
                                ;   preobremenjen).
        rts                     ; 5
```

Podprogram je dolg *81* strojnih ciklov (vštevši ukaz za skok v podprogram). Spremenljivke `I_P3CSR`, `I_TCSR`, `I_TRCSR`, `I_P2DR` so *8*-bitne.

5.4 Omogočanje prekinitev

Podprogram vzpostavi stanje pred onemogočenjem prekinitev:

```
                                ; 6 Ciklov za klic podprograma.
ENAINTS ldaa _P3CSR             ; 3 Vzpostavi staro stanje IS3.
        anda #%10111111         ; 2
        oraa I_P3CSR            ; 4
        staa _P3CSR             ; 3
        ldaa _TCSR              ;   Vzpostavi staro stanje
        anda #%10111111         ;   ICF in TOF.
        oraa I_TCSR             ;
        staa _TCSR              ;
        ldaa _TRCSR             ;   Vzpostavi staro stanje SCI.
        anda #%11101011         ;
        oraa I_TRCSR            ;
        staa _TRCSR             ;
        ldaa _P2DR              ;   Vzpostavi staro stanje IRQ1
        anda #%11111011         ;   (Deluje samo, če je dodano vezje
        oraa I_P2DR             ;   za onemogočanje IRQ1.).
        staa _P2DR              ;
        ldaa #%00000110         ; 2 Prikaži 'I' na 7-segmentnem
        staa $A000              ; 4 prikazovalniku (prekinitve so
                                ;   omogočene).
        rts                     ; 5
```

Podprogram je dolg *65* strojnih ciklov (vštevši ukaz za skok v podprogram).

5.5 Vzpostavitev začetnega stanja sistema

Podprogramu `INIT` dodamo naslednje ukaze, ki vzpostavijo začetno stanje prekinitev in ga shranijo:

```
        ldaa  _P2DDR      ; P22 je izhod.
        oraa  #%00000100  ;
        staa  _P2DDR      ;
        ldaa  _P2DR       ; P22 postavi na '1' (Omogoči IRQ1,
        oraa  #%00000100  ; če je prisotno dodatno vezje.).
        staa  _P2DR       ;
        ....              ; Ostale začetne nastavitve za
        ....              ; prekinitvene podprograme.
        jsr   DISINTS     ; Onemogoči prekinitve in shrani
                          ; stanje.
```

5.6 Prekinitveni podprogrami

Naslednji primer pove skoraj vse, kar je treba vedeti o vključevanju prekinitvenih podprogramov v operacijski sistem RT68:

```
PR1LEN  FDB 200          ;    Rezerviraj 200 ciklov.
                         ; 12 ciklov za shranjevanje stanja
                         ;    procesorja na sklad,
PR1     ldx   #PR1LEN    ;  3 X naj kaže na ta podatek.
        jsr   INTPROL    ;*** Klici uvodni podprogram.
        bcc   PR1OK      ;  3 Je izvajanje prekinitvenega
                         ;    podprograma dovoljeno?
        rti              ; 10 Če ni, zaključi z delom.
PR1OK   ....             ;    --------------------------------
        ....             ;    Sem zapišemo telo prekinitvenega
        ....             ;    podprograma.
        ....             ;    --------------------------------
        rti              ; 10 Konec.
```

Število strojnih ciklov, ki jih želimo rezervirati za prekinitveni podprogram, podamo v ROM-u. Na začetku prekinitvenega podprograma postavimo register X, da kaže na to število in pokličemo podprogram `INTPROL`, katerega dolžina zavisi od načina dodeljevanja procesorskega časa (statično, dinamično). Če je po vrnitvi iz podprograma `C=0`, izvedemo telo prekinitvenega podprograma. V primeru, da je `C=1`, zaključimo z delom (`RTI`).

Naslov prekinitvenega podprograma moramo dodati v tabelo prekinitvenih vektorjev:

```
        ORG   $FFF0       ; Prekinitveni vektorji so na
                          ;   naslovih $FFF0-$FFFF.
        FDB   PR1         ; SCI
        FDB   HALT        ; TOF
        FDB   _OCF        ; OCF (razvrščevalnik)
        FDB   HALT        ; ICF
        FDB   HALT        ; IRQ1, IS3
        FDB   HALT        ; SWI
        FDB   HALT        ; NMI
        FDB   _RES        ; RESET
```

V zgornji tabeli smo nastavili SCI prekinitveni vektor, tako da kaže na naš prekinitveni podprogram. Če ne uporabljamo katere od prekinitev, postavimo pripadajoč prekinitveni vektor na naslov `HALT`. Če se bo slučajno pojavila zahteva po neuporabljeni prekinitvi, se bo procesor ustavil v neskončni zanki. Na ta način lahko ugotovimo neželene vire prekinitev med razvojem programa.

5.7 Statično dodeljevanje procesorskega časa

Definirati moramo konstanti:

```
CYCLIM  EQU   400
CYCMAX  EQU   200
```

Prva določa število strojnih ciklov v eni časovni rezini, ki jih rezerviramo za prekinitvene podprograme, druga pa je enaka dolžini najdaljšega prekinitvenega podprograma. Konstanti uporabljata razvrščevalnik in uvodni podprogram. Namesto *400* in *200* lahko uporabimo tudi drugačni vrednosti. Vsekakor pa mora veljati

$$1200 > CYCLIM \geq CYCMAX \,. \qquad (3)$$

5.7.1 Razvrščevalnik

Za statično dodeljevanje procesorskega časa je potrebno izhodiščni razvrščevalnik minimalno spremeniti:

```
                                ;12 ciklov za shranjevanje
                                ;   stanja procesorja na sklad.
_OCF    ldaa  SCHTST            ; 4 Preberi testni byte.
        beq   SCHOK             ; 3 Če se prejšnja časovna rezina
        ldaa  #%01101101        ;   ni zaključila pravočasno,
        staa  $A000             ;   prikaži 'S' na 7-segmentnem
SCHERR  bra   SCHERR            ;   prikazovalniku in se ustavi
                                ;   v neskončni zanki.
SCHOK   inc   SCHTST            ; 6 Postavi testni byte (Pomeni,
                                ;   da časovna rezina teče.).
        ldaa  _TCSR             ; 3 Briši TOF.
        ldd   _OCR              ; 4 Preberi OCR,
        addd  #1200             ; 4 povečaj ga za dolžino časovne
        std   _OCR              ; 4 rezine in shrani.
        ldd   #CYCLIM           ; 5 Postavi števec rezerviranih
                                ;   strojnih ciklov na začetno
                                ;   vrednost CYCLIM.
        std   CYCLEFT           ; 5
        jsr   ENAINTS           ;65 Omogoči prekinitve.
        ldx   SCHPTR            ; 5 Preberi kazalec na tekoče
        ldx   0,X               ; 5 opravilo.
        cli                     ; 2 Briši prekinitveno masko.
        jsr   0,X               ; 6 Zaženi opravilo.
        ldaa  SCHPTR+1          ; 4 Popravi kazalec na tekoče
        adda  #2                ; 2 opravilo.
        anda  #%00011110        ; 2
        staa  SCHPTR+1          ; 4
        sei                     ; 2 Postavi prekinitveno masko.
        clr   SCHTST            ; 6 Naznači, da smo časovno rezino
                                ;   uspešno zaključili.
        rti                     ;10 Vrni se iz OCF prekinitvenega
                                ;   podprograma.
```

Spremenljivka `CYCLEFT` je *16*-bitna in vsebuje število strojnih ciklov v časovni rezini, ki so na voljo prekinitvenim podprogramom.
Spremenljivka `SCHTST` je *8*-bitna. Če je različna od *0*, se izvaja opravilo iz urnika. Uporabljamo jo za ugotavljanje prekoračitve dolžine časovne rezine.

Izhodiščna različica razvrščevalnika vsebuje napako, ki redko pride do izraza, in jo je zaradi tega težko odkriti. Pred postavitvijo spremenljivke `SCHTST` na *0* je treba onemogočiti prekinitve, saj prekoračitve dolžine časovne rezine, ki se zgodi po brisanju `SCHTST` in pred ukazom `RTI`, ne zaznamo. Če tega ne bi storili, bi se pri daljšem delovanju slabo napisanega programa (predolga opravila) kopičila gnezdenja TOF prekinitev, kar bi imelo za posledico prekoračitev sklada in nepredvidljivo obnašanje sistema.

Razvrščevalnik se izvaja *163* strojnih ciklov in porabi *14%* procesorskega časa. Za opravila iz urnika in glavni program ostaja torej *1037* strojnih ciklov na časovno rezino.

5.7.2 Uvodni podprogram

Uvodni podprogram je treba poklicati na začetku vsakega prekinitvenega podprograma. V register X postavimo naslov, kjer je shranjeno število strojnih ciklov, ki jih rezerviramo za prekinitveni podprogram v obliki nepredznačenega *16*-bitnega števila. Če ni dovolj strojnih ciklov za izvajanje prekinitvenega podprograma, je ob vrnitvi bit C postavljen na '1'.

Pri statičnem dodeljevanju procesorskega časa uvodni podprogram beleži rezervirane strojne cikle za prekinitveni podprogram. Ko preostane premalo strojnih ciklov za izvrševanje najdaljšega prekinitvenega podprograma, onemogoči prekinitve do nastopa naslednje časovne rezine. Čeprav se ob pravilni nastavitvi konstant `CYCLIM` in `CYCMAX` ne more zgoditi, da bi ciklov zmanjkalo že za odgovor na prvo zahtevo po prekinitvi znotraj časovne rezine, je vseeno vključeno dodatno preverjanje, ki naredi sistem odpornejši na programerske napake.

```
                          ; 6 ciklov za klic podprograma.
INTPROL ldd   CYCLEFT     ; 5 Preberi CYCLEFT.
        subd  0,X         ; 6 Odštej rezervirane cikle.
        bcc   IOK         ; 3 Če ostane manj kot 0 ciklov (C=1),
        jsr   DISINTS     ;81   onemogoči prekinitve.
        rts               ; 5 Vrni se iz podprograma.
IOK     std   CYCLEFT     ; 5 Popravi CYCLEFT.
        subd  #CYCMAX     ; 6 Preveri, če je ostalo dovolj
                          ;   ciklov za najdaljši prekinitveni
                          ;   podprogram (CYCMAX).
        bcc   PROLOK      ; 3 Če ne, onemogoči prekinitve.
        clc               ; 2 Dovoli izvajanje prekinitvenega
                          ;   podprograma.
        jsr   DISINTS     ;81
PROLOK  rts               ; 5 Vrni se iz podprograma.
```

V najslabšem primeru traja izvajanje podprograma (vštevši ukaz za skok v podprogram) *122* strojnih ciklov.

5.8 Dinamično dodeljevanje procesorskega časa

Definirati moramo konstanti:

```
CYCTOP  EQU   4000
CYCBOT  EQU   3500
```

CYCTOP določa največji zaostanek zaradi podaljševanja časovnih rezin. Če je to število preseženo, so prekinitve onemogočene, dokler zaradi krajšanja časovnih rezin zaostanek ne pade pod CYCBOT. Veljati mora:

$$CYCTOP > CYCBOT \tag{4}$$

5.8.1 Zagon razvrščevalnika

Izhodiščni različici podprograma dodamo na koncu ukaz, ki omogoči prekinitve.

```
        jsr   ENAINTS
```

Podprogramu za vzpostavitev začetnega stanja sistema (5.5) dodamo naslednji vrstici:

```
        ldd   #$0000
        std   CYCEXT
```

S tem postavimo skupen zaostanek na *0*. Spremenljivka CYCEXT je nepredznačeno *16*-bitno število.

5.8.2 Razvrščevalnik

Za razliko od statičnega dodeljevanja procesorskega časa, je razvrščevalnik tokrat precej daljši:

```
                              ;12 ciklov za shranjevanje
                              ;   stanja procesorja na sklad.
_OCF    ldaa  SCHTST          ; 4 Preberi testni byte.
        beq   SCHOK           ; 3 Če se prejšnja časovna rezina
        ldaa  #%01101101      ;   ni zaključila pravočasno,
        staa  $A000           ;   prikaži 'S' na 7-segmentnem
SCHERR  bra   SCHERR          ;   prikazovalniku in se ustavi
                              ;   v neskončni zanki.
SCHOK   inc   SCHTST          ; 6 Postavi testni byte (Pomeni,
                              ;   da časovna rezina teče.).
        ldaa  _TCSR           ; 3 Briši TOF.
        ldd   _OCR            ; 4 Preberi OCR,
        addd  #1200           ; 4 povečaj ga za dolžino časovne
        std   _OCR            ; 4 rezine in shrani.
        ldx   SCHPTR          ; 5 Preberi kazalec na tekoče
        ldx   0,X             ; 5 opravilo.
        cli                   ; 2 Briši prekinitveno masko.
        jsr   0,X             ; 6 Zaženi opravilo.
        ldaa  SCHPTR+1        ; 4 Popravi kazalec na tekoče
        adda  #2              ; 2 opravilo.
        anda  #%00011110      ; 2
        staa  SCHPTR+1        ; 4
        sei                   ; 2 Postavi prekinitveno masko.
        clr   SCHTST          ; 6 Naznači, da smo časovno rezino
```

```
                                ;   uspešno zaključili.
        ldd   _FRC              ; 4 Preberi FRC.
        tstb                    ; 2 Preveri, če je prišlo do prenosa.
        bne   SFRCOK            ; 3 Ni prenosa.
        ldd   _FRC              ; 4 Ponovi branje.
        bra   SSHORT            ; 3 Branje opravljeno.
SFRCOK  addd  #$0009            ; 4 Popravi odčitek in porabi 7
        bra   SSHORT            ; 3 strojnih ciklov.
                                ; Sistemski cikli:
                                ;     4  cikli od branja do SSHORT
                                ;   172  ciklov od SSHORT do rti
                                ;     5  ciklov za vsak primer.
                                ;  -----
                                ;   181  ciklov največ
SSHORT  addd  #181              ; 4 Dodaj sistemske cikle.
        std   TMPFRC            ; 5 Shrani v TMPFRC.
        ldd   _OCR              ; 4 Preberi OCR.
        subd  TMPFRC            ; 6 Odštej TMPFRC. Rezultat je
        std   SLICES            ; 5 število ciklov, za katere bomo
        ldd   CYCEXT            ; 5 skrajšali časovno rezino.
        subd  SLICES            ; 6 Odštej SLICES od CYCEXT.
        bcs   CLIP              ; 3 Če SLICES ni večji od CYCEXT
        std   CYCEXT            ; 5 popravi CYCEXT
        bra   SHORTEN           ; 3 Skrajšaj časovno rezino.
CLIP    ldd   CYCEXT            ; 5 SLICES je večji od CYCEXT;
        std   SLICES            ; 5   postavi SLICES=CYCEXT.
        ldd   #$0000            ; 3 CYCEXT postavi na 0
        std   CYCEXT            ; 5 (Zaostanek je nadoknaden.).
SHORTEN ldd   _OCR              ; 4 Krajšanje časovne rezine.
        subd  SLICES            ; 6 Odštej SLICES od OCR.
        std   _OCR              ; 4 Shrani rezultat v OCR.
        ldd   CYCEXT            ; 5
        subd  CYCBOT            ; 6 Če je CYCEXT<CYCBOT
        bcc   NOSHORT           ; 3
        jsr   ENAINTS           ;65 omogoči prekinitve.
NOSHORT rti                     ;10 Vrni se iz OCF prekinitvenega
                                ;   podprograma.
```

Razvrščevalnik se v najslabšem primeru izvaja *261* strojnih ciklov, kar znese *22%* procesorskega časa.

5.8.3 Uvodni podprogram

Uvodni podprogram skrbi za daljšanje časovnih rezin. Uporabljamo ga na enak način, kot je bilo povedano v 5.7.2.

```
                                ; 6 ciklov za klic podprograma.
INTPROL ldd  CYCEXT             ; 5 Dodaj rezervirane cikle k
        addd 0,X                ; 6 zakasnitvi in
        std  CYCEXT             ; 5 shrani rezultat v CYCEXT.
        subd #CYCTOP            ; 4 Primerjaj CYCEXT in CYCTOP.
        bls  INTOK              ; 3 CYCEXT>CYCTOP
        jsr  DISINTS            ;81   onemogoči prekinitve
INTOK   ldaa _TCSR              ; 3 Preberi TCSR (briši OCF).
        ldd  _OCR               ; 4 Podaljšaj časovno rezino.
        addd 0,X                ; 6
        std  _OCR               ; 4
        clc                     ; 2 Dovoli izvajanje prekinitvenega
                                ;   podprograma.
```

```
            rts                 ; 5 Vrni se iz podprograma.
```

Podprogram je dolg (vštevši ukaz za skok v podprogram) *134* strojnih ciklov.

5.9 Gnezdenje prekinitev in opravila, ki ne smejo biti prekinjena

V nekaterih primerih je zaželeno gnezdenje prekinitev. Gnezdenje prekinitev omogočimo z brisanjem globalne prekinitvene maske z ukazom `CLI`, ki ga izvedemo v prekinitvenem podprogramu. S tem dovolimo, da nova zahteva po prekinitvi sproži pripadajoč prekinitveni podprogram že med izvajanjem tekočega prekinitvenega podprograma.

Znotraj prekinitvenega podprograma smemo omogočiti prekinitve šele ko se je končalo izvajanje uvodnega podprograma. Tako nadzor nad prekinitvami ostane v rokah operacijskega sistema.

V primeru, da imamo opravilo, ki ne sme biti prekinjeno, postavimo na začetek opravila ukaz `SEI`, na konec pa ukaz `CLI`, kar onemogoči prekinitve za čas izvajanja opravila. Taki opravili sta na primer generiranje impulza določene širine in komunikacija preko I^2C vodila.

6 Primerjava obeh pristopov

Dodeljevanje procesorskega časa	Brez	Statično	Dinamično
Dolžina uvodnega podprograma	-	*122*	*134*
Dolžina razvrščevalnika	*88*	*163*	*261*
Poraba procesorskega časa	*8%*	*14%*	*22%*
Čas, ki preostane za izvajanje opravila, glavnega programa in prekinitvenih podprogramov v časovni rezini	*1112 (92%)*	*1037 (86%)*	*939 (78%)*
Čas od zahteve po prekinitvi do začetka izvajanja telesa prekinitvenega podprograma (ko je I=0)	-	*140 (0,114ms)*	*152 (0,124ms)*

Tabela 1: Primerjava obeh pristopov. Če ni posebej označeno, pomenijo številke strojne cikle mikrokrmilnika.

Obe predlagani rešitvi bistveno ne zapleteta razmer v operacijskem sistemu. Temu v prid govori dejstvo, da ostaja uporaba prekinitvenih podprogramov za uporabnika večinoma enaka, kot bi bila v sistemu z enim samim opravilom. Jedro operacijskega sistema je ostalo pregledno, kratko in s tem tudi razumljivo uporabnikom, ki se s sistemi v realnem času šele seznanjajo.

Razvrščevalnik je pri dinamičnem dodeljevanju procesorskega časa najdaljši. To je bilo tudi za pričakovati, saj je ta pristop dokaj zapleten v primerjavi s statičnim dodeljevanjem procesorskega časa. Poraba procesorskega časa je razen od dolžine razvrščevalnika odvisna tudi od dolžine časovne rezine. Da bi bila izraba procesorskega časa boljša, bi bilo potrebno podaljšati časovno rezino. Posledica bi bila bolj groba delitev časa. Če bi želeli ohraniti enako porabo procesorskega časa ob enakem številu časovnih rezin v enoti časa (*1024*), bi morali za statično dodeljevanje procesorskega časa uporabiti *2x*, za dinamično pa *3x* hitrejši mikrokrmilnik.

Glede hitrosti odziva na zahtevo po prekinitvi sta oba pristopa enakovredna. Če zahteva po prekinitvi nastopi medtem, ko je postavljena globalna prekinitvena maska, se prekinitveni podprogram začne izvajati šele po brisanju prekinitvene maske.

Število zahtev po prekinitvi, ki jih je sistem sposoben obdelati v enoti časa, lahko najlepše ilustriramo s primerom. Če pri statičnem dodeljevanju procesorskega časa rezerviramo *500* strojnih ciklov v časovni rezini za prekinitvene podprograme in je prekinitveni podprogram dolg *250* strojnih ciklov, se lahko izvedeta *2* prekinitvena podprograma na časovno rezino, kar znese *2048* izvajanj v sekundi. To zadostuje za upravljanje s serijskim vmesnikom mikrokrmilnika pri hitrosti *19200bps* (*8* podatkovnih bitov, *1* stop bit).

Pri dinamičnem dodeljevanju procesorskega časa je slika kompleksnejša. Če največjo zakasnitev nastavimo na *4000* strojnih ciklov in je prekinitveni podprogram dolg *250* strojnih ciklov, je sistem sposoben obdelati *16* zahtev po prekinitvi, ki nastopijo ena za drugo. To nanese *4915* izvajanj v sekundi. Dolgoročno povprečje je seveda manjše. Odvisno je od tega, koliko procesorskega časa ostane glavnemu programu. Če je vsako opravilo dolgo *500* strojnih ciklov, ostane za glavni program *1200-500-261=439* strojnih ciklov na časovno rezino. Prekinitveni podprogram se lahko izvede *439·1024/250=1798* krat v sekund, kar zadostuje za upravljanje s serijskim vmesnikom mikrokrmilnika pri hitrosti *16800bps* (*8* podatkovnih bitov, *1* stop bit).

Dinamično dodeljevanje procesorskega časa ima to prednost pred statičnim, da je sistem sposoben kratkoročno porabiti ves procesorski čas za odzivanje na prekinitvene podprograme. Cena za to je, da ne moremo več določiti začetkov časovnih rezin. Statično dodeljevanje procesorskega časa bi uporabili v primerih, ko je nujno, da se časovne rezine začenjajo ob vnaprej znanih časih. (na primer za generiranje verige impulzov). Da to dosežemo, moramo dovoliti gnezdenje prekinitev s čimer opravilom iz urnika damo prednost pred prekinitvenimi podprogrami.

Operacijski sistem smo preizkusili z naslednjim prekinitvenim podprogramom, ki se je izvedel vsakič, ko je bila sprejeta beseda preko serijskega komunikacijskega vmesnika oziroma je nastopila napaka prekrivanja ali napaka okvirja:

```
ISCILEN FDB 200          ;    Rezerviraj 200 strojnih ciklov
                         ;    za prekinitveni podprogram
                         ; 12 ciklov za shranjevanje stanja
                         ;    procesorja na sklad.
ISCI    ldx   #ISCILEN   ;  3
        jsr   INTPROL    ;*** Skok v uvodni podprogram
        bcc   ISCIOK     ;  3 Preveri, če je dovoljeno izvajanje
                         ;    prekinitvenega podprograma.
        rti              ; 10 Če ne, zaključi z delom.
ISCIOK  ldaa _TRCSR      ;  3 Preberi stanje vmesnika
        ldab _RDR        ;  3 Preberi sprejeto besedo
        bita #%01000000  ;  2 Preveri, če je prišlo do napake.
        beq  ISCIECH     ;  3
        ldx  CNTERR      ;  5 Če je, povečaj števec napak.
        inx              ;  3
        stx  CNTERR      ;  5
ISCIECH ldx  CNTCHAR     ;  5 Povečaj števec znakov.
        inx              ;  3
        stx  CNTCHAR     ;  5
        ldab #42         ;  2
        stab _TDR        ;  3 Odgovori z znakom '*'.
        rti              ; 10 Konec prekinitvenega podprograma.
```

Na vseh *16* mest v urniku smo postavili isto opravilo, katerega dolžino smo lahko spreminjali v korakih po *6* strojnih ciklov.

Pri statičnem dodeljevanju procesorskega časa smo rezervirali *200* strojnih ciklov na časovno rezino za izvajanje prekinitvenih podprogramov. Za prekinitveni podprogram smo prav tako rezervirali *200* strojnih ciklov. Nato smo iz osebnega računalnika prenašali datoteko preko serijskega vmesnika v sistem RT68-01 (*9600bps*, *8* podatkovnih bitov, *1* stop bit). Dolžino opravil v urniku smo spreminjali, dokler sistem tekom prenosa ni javil prekoračitve dolžine časovne rezine. To se je zgodilo, ko so bila opravila v urniku dolga najmanj *866* strojnih ciklov. Po naši napovedi, bi se to moralo zgoditi, ko bi opravilo v urniku postalo daljše od *1037-200=837* strojnih ciklov. Sistem je »zdržal« nekoliko več, ker so vse dolžine vzete za najslabši primer (za najdaljše možno izvajanje podprogramov). Če sistem ni javil prekoračitve časovne rezine, je bila datoteka prenešena brez napak.

Pri dinamičnem dodeljevanju procesorskega časa smo največjo zakasnitev nastavili na *4000* strojnih ciklov. Zaradi večje dolžine uvodnega podprograma smo za prekinitveni podprogram rezervirali *210* strojnih ciklov. Podatke smo prenašali z enako hitrostjo, kot v prejšnjem

primeru. Dolžino opravil v urniku smo spreminjali, dokler sistem ni postal preobremenjen in so se pri prenosu pojavile napake. To se je zgodilo, ko je bila dolžina opravila v urniku večja ali enaka *794* strojnih ciklov. Če upoštevamo, da je prenos pri *9600bps* zahteval v povprečju $960 \cdot 210/1024=196,9$ strojnih ciklov na časovno rezino, bi moral sistem postati preobremenjen pri dolžini opravila večji od $939-197=742$ strojnih ciklov. Razliko lahko spet pripišemo dejstvu, da smo v izračunu uporabili vrednosti za najslabši možni primer.

7 Sklep

Kljub strogim omejitvam, ki jih postavlja relativno majhna hitrost mikrokontrolerja M6803, nam je uspelo dodati prekinitve v operacijski sistem v realnem času RT68. Da smo nalogo lahko opravili, smo se morali sprijazniti z nekaterimi omejitvami in poenostavitvami. Tako je nekatere stvari (na primer določanje dolžine prekinitvenega podprograma), ki bi jih na zmogljivejši platformi opravil sam procesor, odgovoren programer. Opustili smo tudi strukture za preprečevanje hkratnega dostopa do skupnih enot (semafor, monitor).

Obravnavali smo dva pristopa za uvedbo prekinitev: statično in dinamično dodeljevanje procesorskega časa. Oba sta se izkazala kot dovolj učinkovita za uporabo v novi različici operacijskega sistema.

Statično dodeljevanje procesorskega časa je primerno za sisteme, v katerih je nujno, da za vsako opravilo natanko vemo, kdaj se bo izvedlo.

Dinamično dodeljevanje procesorskega časa je še posebej učinkovito v sistemih, kjer je potrebno obravnavati veliko število zahtev po prekinitvi, ki se pojavijo v krajšem času med daljšimi premori.

Spremenjen operacijski sistem smo preizkusili in ugotovili, da se obnaša tako, kot smo na začetku napovedali. To potrjuje pravilnost izvedbe obeh pristopov.

Glede na to, da je vsako nadaljnje nadgrajevanje operacijskega sistema RT68 povezano z manjšanjem časa, ki je na voljo za izvajanje opravil, bi bilo primerno razmisliti o novi inačici učnega mikrokrmilniškega sistema, ki bi bila zgrajena okoli zmogljivejšega mikrokrmilnika. Pri tem bi morali spet poiskati kompromis med zmogljivostjo mikrokrmilnika in njegovo kompleksnostjo. Ocenili smo, da bi bil potreben vsaj *3x* hitrejši mikrokrmilnik.

8 Priloga: preprečevanje hkratnega dostopa v operacijskem sistemu brez podpore prekinitvam

```
;_____________________ SINHRONIZACIJA Z RAZVRŠČEVALNIKOM _____________________
;  Pokliči ta podprogram iz kateregakoli dela programa, ki se ne izvaja v
;  realnem času, da dobiš D neprekinjenih strojnih ciklov. Ob vrnitvi iz
;  podprograma imaš na voljo zahtevano število strojnih ciklov in še D ciklov
;  do naslednjega izvajanja razvrščevalnika.
;  Z D=0 in klicanjem podprograma dobiš število strojnih ciklov do naslednjega
;  izvajanja razvrščevalnika.
;  Primer:
;    ldd   #100           ; 3 Zahtevamo 100 neprekinjenih strojnih ciklov
;    jsr   SCHSYN         ; 6
;                         ;   Od tod naprej je na voljo 100+D neprekinjenih
;                         ;     strojnih ciklov
;-----------------------------------------------------------------------------
SCHSYN  addd  #$0020      ; 4 Dodaj 32 strojnih ciklov, ki jih bo zase porabil
                          ;   podprogram SCHSYN.
        std   SYNCYC      ; 5 Shrani rezultat v SYNCYC.
SCHWAI  ldd   _OCR        ; 4 Preberi primerjalni register prostotekočega
                          ;     števca (OCR)
        subd  SYNCYC      ; 6 Odštej zahtevano število strojnih ciklov.
        pshb              ; 3 Shrani D na skladu.
        psha              ; 3
                          ; ----//// Branje prostotekočega števca (FRC)
        ldd   _FRC        ; 4 Preberi FRC
        tstb              ; 2 Preveri, če je prišlo do prenosa v nižjem bytu
        bne   SYNOK       ; 3 Ni prenosa, odčitek je pravilen
        ldd   _FRC        ; 4 Ponovno preberi FRC
        bra   SYNCORR     ; 3 Branje končano
SYNOK   addd  #$0009      ; 4 - Če je že prvo branje bilo uspešno, popravi
        bra   SYNCORR     ; 3 -   odčitek in izenači število strojnih ciklov.
                          ; --- Odčitek FRC je za 4 manjši od vrednosti, ki jo
                          ;       bo FRC vseboval v 1. ciklu ukaza na
                          ;       naslovu SYNCORR.
                          ; ----//// Konec branja FRC
SYNCORR std   SYNFRC      ; 5 Shrani FRC v SYNFRC
        pula              ; 4 Preberi D s sklada
        pulb              ; 4
        subd  SYNFRC      ; 6 Odštej od D odčitek FRC
        bcs   SCHWAI      ; 3 Če je rezultat manjši od 0, se vrni na SCHWAI.
        rts               ; 5 Vrni se iz podprograma
                          ; --- Cikli, ki jih porabi SCHSYN zase:
                          ; ---   1+3          (od branja FRC do SYNCORR)
                          ; ---  +5+4+4+6+3+5  (od SYNCORR do prve operacije
                          ; ---                 po rts)
                          ; ---  +1            (za vsak primer)
                          ; ---  =32
```

9 Viri

[1] D. Kodek, *Mikroprocesorji - delovanje in uporaba*, ZOTKS, Ljubljana, 1986.

[2] J. E. Cooling, S*oftware Design for Real-time Systems*, Chapman and Hall, London, 1991.

[3] M. F. zaletel, »Učni mikroprocesorski sistem«, *diplomska naloga*, Univerza v Ljubljani, FER, 1995.

[4] T. Rehar, »Učni mikroprocesorski sistem«, *diplomska naloga*, Univerza v Ljubljani, FER, 1995.

[5] S. Levač, »Nova inačica učnega mikrokrmilniškega sistema«, *diplomska naloga*, Univerza v Ljubljani, FE, 1997.

[6] M. Sajko, »Operacijski sistem v realnem času za učni mikroprocesorski modul«, *diplomska naloga*, Univerza v Ljubljani, FER, 1994.

[7] T. Tuma, *Mikrokrmilniški sistemi - programiranje v realnem času*, Založba FE in FRI, Ljubljana, 1997.

[8] D. M. Auslander, *Real-time Software for Control: Program Examples in C*, Prentice Hall, New Jersey, 1989.

IZJAVA

Izjavljam, da sem diplomsko delo izdelal samostojno pod vodstvom mentorja doc. dr. Tadeja Tume, dipl. ing.. Izkazano pomoč drugih sodelavcev sem v celoti navedel v zahvali.