

# **Mikroračunalniki v močnostni elektroniki**

*Vanja Ambrožič*

Fakulteta za elektrotehniko

Ljubljana, 1999

## **PREDGOVOR**

*Luki*

Ta učbenik je nastal iz beležk za predavanja pri predmetu Mikroračunalniško vodenje tehnoloških procesov, ki sedaj nosi naziv Vodenje tehnoloških procesov. Oba predmeta sta bila zamišljena kot seznanjanje študentov končnega letnika (četrtega po starem programu ozziroma petega po novem programu) Sistemsko-tehnološke smeri na Fakulteti za elektrotehniko z vlogo mikroračunalnikov v vezjih močnostne elektronike. Študenti te smeri poznajo osnove arhitekture mikroprocesorjev že iz predhodno poslušanega sorodnega predmeta, zato je ta učbenik zamišljen kot nadgradnja že doseženih znanj. Celotno gradivo ima za izhodišče specifične aplikacije mikroprocesorjev na področju močnostne elektronike, kot so npr. električni pogoni, pretvorniška vezja, aktivni močnotni filtri itd., pri čemer topologija takšnih vezij v tem učbeniku ni posebej obravnavana.

Pri dosedanjem pedagoškem delu sem večkrat ugotovil, da imajo študenti predsodke o uporabi mikroprocesorjev: mikroračunalniki, ki ne delajo s frekvenco nekaj sto MHz, nimajo matematičnega koprocesorja, nekaj deset Mbytov RAM pomnilnika ali nekaj Gbytov trdega diska, so neuporabni!

Ena od nalog inženirja je prav iskanje optimalnega kompromisa med zmogljivostmi uporabljenih komponent in končno ceno primernega mikroračunalniškega sistema ter časom za razvoj.

Ta učbenik bo upravičil svoj obstoj, če bo študentom pomagal pri vpogledu v ozadje opisanih problemov in jih stimuliral k previdnejši uporabi vsakdanjih fraz.

Ob tej priložnosti se bi rad zahvalil vsem, ki so me pri delu spodbujali. Posebna zahvala velja as. dr. Davidu Nedeljkoviću, univ. dipl. inž. el., katerega strokovne in slovnične pripombe so veliko prispevale k nastajanju tega učbenika.

*Avtor*

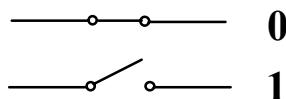
# 1. ŠTEVILSKI SISTEMI

## 1.1 Številske enote

### 1.1.1 Bit

Obdelava podatkov v mikroprocesorski tehniki sloni na *dvojiškem (binarnem)* številskem sistemu. Takšen sistem je izbran zato, ker ga je tehnično najlažje realizirati. Dve ločeni in edini možni stanji zelo enostavno ustvarimo s stikalnim elementom (slika 1. 1):

- stikalo zaprto (kontakta sklenjena, tok teče),
- stikalo odprtlo (kontakta razklenjena, tok ne teče).



*Slika 1. 1: Določanje dveh stanj s pomočjo stikala*

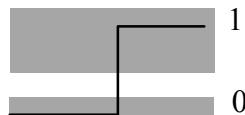
V praksi so stikala realizirana s polprevodniškimi elementi (tranzistorji in diodami) ter pasivnimi elementi (upori in kondenzatorji). Sodobni stikalni elementi so navadno izdelani v MOS (npr. HCMOS) tehnologiji. Uporabljeni tehnologiji nanašanja različnih polprevodniških plasti omogoča njihovo fizično realizacijo na izredno majhni površini (npr. nekaj stotisoč tranzistorjev na čipu), kjer se debeline plasti merijo v mikronih ( $10^{-6}$  m). Na ta način je možno na enoto površine spraviti veliko število podobnih stikal, ki preklaplja z izredno visoko frekvenco (nekaj deset ali celo nekaj sto MHz) in imajo izredno majhno porabo (npr. nekaj sto  $\mu$ W v mirujočem, "standby" režimu).

Dve stanji s slike 1. 1 lahko ponazorimo s številoma 0 in 1 (ali "L" in "H": angl. low in high). Kateremu izmed njiju bomo dodelili posamezno številsko vrednost, je stvar dogovora. S tem smo določili najmanjšo enoto odločanja ("DA" in "NE", ali "RESNIČNO" in "NERESNIČNO" - angl. "TRUE" in "FALSE"). Za zapis obeh možnosti potrebujemo le eno dvojiško števko - *digit* (latinska beseda za prst). Od tod tudi ime najmanjšega nosilca digitalne informacije: *bit* (angl. **binary digit**)<sup>1</sup>.

V praksi realiziramo binarna stanja z napetostnimi ali tokovnimi signalni različnih nivojev. Ti so posledica preklopnih stanj tranzistorja. Z ozirom na tehnološko realizacijo mikroelektronskega vezja ter njegovo konkretno aplikacijo razlikujemo več standardov, ki določajo višino nivoja za obe stanji (slika 1. 2). Najbolj razširjen je TTL standard, ki določa nazivno napetost 0 V za signal 0 in 5 V za signal 1. V praksi je tako strogo definirane nivoje nemogoče doseči zaradi motenj in napetostnih padcev, zato sta za obe stanji predvidena

<sup>1</sup> Tukaj imamo opravka z besedno igro, saj *bit* v angleščini pomeni tudi košček, malenkost.

pasova (tabela 1.1). Če zaradi kateregakoli razloga napetosti ali tokovi zasedejo vmesne vrednosti, logično stanje ostaja nedoločeno.



**Slika 1. 2: Pasova za določanje nivojev logičnih stanj 0 in 1**

| Tehnologija            | $U_{IH}$ [V]      | $U_{IL}$ [V]      | $U_{OH}$ [V]     | $U_{OL}$ [V] |
|------------------------|-------------------|-------------------|------------------|--------------|
| Standardna TTL         | > 2,0             | < 0,8             | > 2,4            | < 0,4        |
| S-TTL                  | > 2,0             | < 0,8             | > 2,7            | < 0,5        |
| LS-TTL                 | > 2,0             | < 0,8             | > 2,7            | < 0,5        |
| CMOS                   | 4,0 - 8,0         | 1,0 - 2,0         | 4,5 - 9          | < 0,5        |
| HCMOS ( $U_S = 4,5$ V) | $> 0,7 \cdot U_S$ | $< 0,3 \cdot U_S$ | $> (U_S - 0,5V)$ | < 0,4        |

### Pojasnilo:

$U_{IH}$ : vhodna napetost v logično vezje, logični nivo 1

$U_{IL}$ : vhodna napetost v logično vezje, logični nivo 0

$U_{OH}$ : izhodna napetost iz logičnega vezja, logični nivo 1

$U_{OL}$ : izhodna napetost iz logičnega vezja, logični nivo 0

**Tabela 1.1: Logični nivoji za različne logične družine [3]**

V programirljivih krmilnikih oz. industrijskih procesorjih se pogostokrat uporablja nivoja 0 V in 24 V za napetostne signale ali 0 mA in 20 mA za tokovne signale (npr. pri krmilnikih Simatic tvrdke SIEMENS [4] itd.

Omenimo še to, da dodelitev višje napetosti ali toka signalu 1 velja za t.i. "pozitivno logiko". Pri "negativni logiki" ustreza višji napetosti signal 0.

### 1.1.2 Nibble

Število različnih stanj, ki jih zapišemo z enim samim bitom, je omejeno na le dve. Zato je smiselno bite povezati v večje enote (tipe, formate) in na ta način povečati število možnih stanj<sup>2</sup>. Podatek, ki sestoji iz dveh bitov, lahko doseže štiri možna stanja, tribitni pa osem (tabela 1.2).

<sup>2</sup> Podobno je pri decimalnih številih: ena števka (enice) omogoča le 10 različnih stanj (od 0 do 9), dve števki (desetice in enice)  $100 = 10^2$  (0 - 99), tri (stotice, desetice in enice)  $1000 = 10^3$  (0 - 999) itd.

| 1 bit | 2 bita | 3 biti |
|-------|--------|--------|
| 0     | 00     | 000    |
| 1     | 01     | 001    |
|       | 10     | 010    |
|       | 11     | 011    |
|       |        | 100    |
|       |        | 101    |
|       |        | 110    |
|       |        | 111    |

**Tabela 1.2: Enobitne, dvobitne in tribitne enote**

Sedaj že lahko govorimo o *ločljivosti* (angl. resolution), ki je določena z izrazom  $1/2^n$  kjer je  $n$  število bitov. Zaenkrat se bomo ustavili pri formatu štirih bitov ( $2^4 = 16$ ), ki ga imenujemo *nibble* (dobesedno angl. grižljaj)<sup>3</sup>.

Binarnemu zapisu lahko dodelimo tudi kvantitativen pomen z *binarno* ali *dvojiško kodo* (angl. binary code; indeks BIN). Baza tega sistema je 2 analogno *decimalni* ali *desetiški kodi* (angl. decimal; indeks DEC)<sup>4</sup>, z bazo 10, ki jo uporabljamo v vsakdanjem življenju.

Na primeru nibbla s slike 1.4 vidimo način interpretacije binarnega podatka v decimalni kodi. Skrajnji desni bit ima utež 1 ( $2^0$ ), naslednji 2 ( $2^1$ ), največjo pa ima skrajnji levi bit (v tem primeru  $2^3 = 8$ ). Z dodajanjem bitov na levi se povečuje njihova utež in s tem tudi maksimalna vrednost celotnega števila. Biti z najmanjšo utežjo (skrajne desni) in največjo utežjo (skrajni levi) označujemo - ne glede na število bitov v podatku - s kraticami *LSB* (angl. Least Significant Bit - najmanj pomemben bit) in *MSB* (angl. Most Significant Bit - najbolj pomemben bit)<sup>5</sup>.

$$0111 = 0 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 0 + 4 + 2 + 1 = 7_{\text{DEC}}$$

**Slika 1.3: Pretvorba iz binarne v decimalno kodo**

Maksimalno število, ki ga zapišemo v enem nibblu, je  $1111_{\text{BIN}} = 15_{\text{DEC}}$ . Splošna formula za izračun maksimalnega števila je:

<sup>3</sup> Nibble je le nekakšna neformalna podatkovna enota med bitom in zlogom, ki nam služi pri pretvorbi iz binarne v heksadecimalno kodo, zato tega izraza ponavadi ne uporabljamo.

<sup>4</sup> V nadalnjem tekstu bomo posamezne formate označevali z ustreznimi indeksi. Decimalna števila bomo pisali z indeksom DEC ali pa brez kakšnegakoli indeksa.

<sup>5</sup> Pri označevanju posameznih bitov znotraj nekega formata izhajamo iz njihove uteži: **LSB** označujemo vedno kot ničti bit (npr. LSB naslovnega vodila je **A0**), **MSB** pa je označen z zaporedno številko  $n-1$  (npr. MSB 16-bitnega naslovnega vodila je **A15**). Posebej je treba biti pozoren na možne napake zaradi besedne igre: prvi bit ima indeks 0, šestnajsti pa 15 (v 16-bitnem formatu)!

$$N_{max} = 2^n - 1, \quad (1.1)$$

kjer je  $n$  število bitov formata. Navidezno neskladje med ločljivostjo ( $2^4 = 16_{DEC}$ ) in maksimalnim številom ( $15_{DEC}$ ) pojasnimo s tem, da je manjkajoče šestnajsto veljavno število 0.

Zapis števil v binarni kodi se izkaže kot zelo nepregleden, zlasti ko imamo opravka s 16- ali 32-bitnimi enotami. Zato takšna števila raje interpretiramo s šestnajstiško ali *heksadecimalno kodo* (angl. hexadecimal; kratica HEX), kjer je baza število 16. Že prej smo ugotovili, da lahko z enim nibblom ustvarimo 16 različnih kombinacij (števil). Iz tega sledi, da štiri bite v nibblu nadomestimo z eno šestnajstiško števko. Šestnajstiške števke od 0 do 9 so ekvivalentne decimalnim, nadaljnje števke ( $10_{DEC} - 15_{DEC}$ ) pa označujemo s črkami<sup>6</sup>  $a_{HEX}, b_{HEX}, c_{HEX}, d_{HEX}, e_{HEX}, f_{HEX}$  (glej tudi tabelo 1.3)!

### 1.1.3 Byte (zlog)

Z dodajanjem bitov pridemo do zelo pomembne enote, ki je sastavljena iz osmih bitov: *byte* (bajt ali zlog). Njegova ločljivost je  $2^8 = 256$ , kar pomeni, da je LSB 1/256<sub>DEC</sub> celotnega obsega. Zlog se je udomačil kot osnovna enota, ki jo uporabljamo v mikroprocesorski tehniki (vse višje standardizirane enote so mnogokratniki zloga). MSB (sedmi bit) pri zlogu ima utež  $2^7$ .

Tabela 1.3 kaže različne možne kombinacije enega byta ter njihove interpretacije v šestnajstiški in desetiški kodi<sup>7</sup>. Pri vseh načinih kodiranja je pomembno poudariti, da so kode le različni načini interpretacije istega binarnega števila!

Vzemimo binarno število 1100 0110<sub>BIN</sub>. Najlaže ga pretvorimo v šestnajstiško število: c6<sub>HEX</sub> ( $1100_{BIN} = c_{HEX}$ ,  $0110_{BIN} = 6_{HEX}$ ). Decimalni zapis lahko dobimo bodisi iz binarnega

$$\begin{aligned} 1100\ 0110_{BIN} &= (1 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^2 + 1 \cdot 2^1)_{DEC} = (128 + 64 + 4 + 2)_{DEC} = \\ &= 198_{DEC}, \end{aligned}$$

ali iz šestnajstiškega zapisa

$$c6_{HEX} = (12 \cdot 16^1 + 6 \cdot 16^0)_{DEC} = (192 + 6)_{DEC} = 198_{DEC}.$$

Iz tabele 1.3 in enačbe (1. 1) je razvidno, da je maksimalno decimalno število, ki ga lahko zapišemo z enim zlogom, 255<sub>DEC</sub> ( $11111111_{BIN} = ff_{HEX}$ ).

V tabeli je prikazana tudi t.i. *BCD koda* (angl. Binary Coded Decimal - dvojiško kodirana decimalna števila). Za razliko od ostalih omenjenih kod, kjer vsak levo dodani bit dobi povečano utež, velja ta logika pri BCD kodi le znotraj enega nibbla, in sicer le za njegove decimalne ekvivalente.

<sup>6</sup> Načeloma lahko te števke šestnajstiških števil zapišemo bodisi z velikimi ali malimi črkami.

<sup>7</sup> Na sliki sta v binarnem zapisu nibbla ločena le zaradi lažje interpretacije. Pri normalnem zapisu binarnega števila tega presledka ni.

| <b>BIN</b><br>binary<br>(binarno) | <b>DEC</b><br>decimal<br>(decimalno) | <b>HEX</b><br>hexadecimal<br>(heksadecimalno) | <b>BCD</b><br>Binary Coded Decimal system<br>(binarno zakodirani decimalni sistem) |
|-----------------------------------|--------------------------------------|---|--|
| 0000 0000                         | 0                                    | 00  | 0000 0000  |
| 0000 0001                         | 1                                    | 01  | 0000 0001  |
| 0000 0010                         | 2                                    | 02  | 0000 0010  |
| 0000 0011                         | 3                                    | 03  | 0000 0011  |
| 0000 0100                         | 4                                    | 04  | 0000 0100  |
| 0000 0101                         | 5                                    | 05  | 0000 0101  |
| 0000 0110                         | 6                                    | 06  | 0000 0110  |
| 0000 0111                         | 7                                    | 07  | 0000 0111  |
| 0000 1000                         | 8                                    | 08  | 0000 1000  |
| 0000 1001                         | 9                                    | 09  | 0000 1001  |
| 0000 1010                         | 10                                   | 0a  | 0001 0000  |
| 0000 1011                         | 11                                   | 0b  | 0001 0001  |
| 0000 1100                         | 12                                   | 0c  | 0001 0010  |
| 0000 1101                         | 13                                   | 0d  | 0001 0011  |
| 0000 1110                         | 14                                   | 0e  | 0001 0100  |
| 0000 1111                         | 15                                   | 0f  | 0001 0101  |
| 0001 0000                         | 16                                   | 10  | 0001 0110  |
| ...                               | ...                                  | ...   | ...  |
| 0111 1111                         | 127                                  | 7f  | 0001 0010 0111   |
| 1000 0000                         | 128                                  | 80  | 0001 0010 1000   |
| 1111 1111                         | 255                                  | ff  | 0010 0101 0101   |

Tabela 1.3: Števila v zlogu v dvojiški, desetiški, šestnajstiški in BCD kodri

**Primer:**

Pri pretvorbi števila

$$1100 \ 0110_{\text{BIN}} = 198_{\text{DEC}}$$

v BCD izhajamo iz njegove decimalne kode. Vsaki decimalni števki bomo privedili binarno četverico (nibble).

$$1_{\text{DEC}} = 0001_{\text{BCD}}, \ 9_{\text{DEC}} = 1001_{\text{BCD}}, \ 8_{\text{DEC}} = 1000_{\text{BCD}} \text{ oz.}$$

$$1100 \ 0110_{\text{BIN}} = 198_{\text{DEC}} = 1 \ 1001 \ 1000_{\text{BCD}}.$$

BCD kodo bi v decimalnem zapisu lahko interpretirali na naslednji način:

$$(1 \cdot 2^0) \cdot 10^2 + (1 \cdot 2^3 + 1 \cdot 2^0) \cdot 10^1 + (1 \cdot 2^3) \cdot 10^0 = 1 \cdot 10^2 + 9 \cdot 10^1 + 8 \cdot 10^0.$$

Kot vidimo, imamo v enem zapisu opravkahkrati z desetiško in binarno bazo.

### Pazi!

Zapisa v binarni in BCD kodi sta si zelo podobna, saj so števke le enice in ničle, njuna interpretacija pa je popolnoma različna.

Do sedaj smo omenili le najpogosteje uporabljane kode. Nekatere druge kode, kot je npr. Excess-3 BCD koda, se niso širše uveljavile.

Zlog je že dolgo časa referenčna enota, npr. za določanje kapacitete pomnilnika z mnogokratnikom zlogov: Kbyte (izgovarjava: kilabajt), Mbyte (megabajt), Gbyte (gigabajt)<sup>8</sup>. Pri tem moramo biti zelo previdni saj te predpone ne označujejo potence števila 10,

$$1 \text{ K} = 10^3 = 1000, \quad 1 \text{ M} = 10^6 = 1\,000\,000, \quad 1 \text{ G} = 10^9 = 1\,000\,000\,000,$$

pač pa izhajajo iz potence števila 2:

$$1 \text{ K} = 2^{10} = 1024_{\text{DEC}}, \quad 1 \text{ M} = (2^{10})^2 = 1.048.576_{\text{DEC}}, \quad 1 \text{ G} = (2^{10})^3 = 1.073.741.824_{\text{DEC}}.$$

Torej 64 Kbyte spomina ni 64000 bytov, temveč  $64 \cdot 1024 = 65536$  bytov!

Preden nadaljujemo z ostalimi enotami, omenimo še kodo, ki jo sicer uporabljam za prikaz znakov. *ASCII* (kratica od "American Standard Code for Information Interchange") je 7-bitna koda, ki se pogosto uporablja v računalnikih in komunikacijah. Z njo kodiramo razne vidne znake (črke, števke, ločila...), kakor tudi znake, ki krmilijo izpis ali vsebujejo nekatere dodatne informacije (npr. koda 0d<sub>HEX</sub> označuje CR, angl. carriage return, ki ga uporabljam za potrdilo vnosa).

#### 1.1.4 Beseda

Povečanje ločljivosti z dodajanjem bitov nas pripelje do naslednjega značilne enote, *besede* (angl. word), ki je sestavljena iz dveh zlogov, torej šestnajstih bitov. Maksimalno število, ki ga lahko zapišemo v eni besedi je (glej tudi (1. 2), pri  $n = 16$ ):

$$1111\ 1111\ 1111\ 1111_{\text{BIN}} = \text{ffff}_{\text{HEX}} = 65535_{\text{DEC}} = 110\ 0101\ 0101\ 0011\ 0101_{\text{BCD}}.$$

Ločljivost 16-bitne enote je  $1/2^{16} = 1/65536$ .

#### 1.1.5 Dvojna in štirikratna beseda

Večina najsodobnejših procesorjev (npr. CPU32 v MC68332, ki ga bomo analizirali v nadaljevanju) sloni na 32-bitni enoti ali *dolgi* oz. *dvojni besedi* (angl. long word ali double word). Največje število v dvojiški kodi bo zapisano s dvaintridesetimi enkami, kar bo v šestnajstiskem zapisu ffff ffff<sub>HEX</sub>. Decimalni ekvivalent je 4 294 967 295.

---

<sup>8</sup> To neskladje naj bi rešile nove SI enote: kibi, mebi, gibi [34].

Naslednja enota je dvakrat večja (64 bitov) in jo imenujemo *štirikratna beseda* (angl. quad word). V večini primerov se javlja le kot rezultat množenja dveh dvojnih besed ali kot deljenec v nekaterih operacijah.

Na koncu še enkrat povzemimo vse do sedaj omenjene enote v njihovem izvirnem zapisu:

**1 byte = 8 bit,**  
**1 word = 2 byte = 16 bit,**  
**1 long word = 2 word = 4 byte = 32 bit,**  
**1 quad word = 2 long word = 4 word = 8 byte = 64 bit.**

## 1.2 Nepredznačena in predznačena cela števila

S povečanjem dolžine enot dosežemo večjo ločljivost in maksimalno možno številsko vrednost (npr. 65535 pri 16-bitni enoti). Pri tem smo imeli opravka le s *celimi* (angl. integer) *nepredznačenimi* (angl. unsigned) števili. V tem podoglavlju bomo govorili o načinu ponazoritve in računanja s predznačenimi celimi števili ter o zapisu števila s plavajočo vejico.

Obstaja več možnih načinov za zapis *negativnih* oz. *predznačenih* števil (angl. signed), najbolj razširjena pa je uporaba *dvojiškega komplementa* (angl. two's complement).

### 1.2.1 Izračun dvojiškega komplementa

Postopek formiranja dvojiškega komplementa ali negativne vrednosti nekega števila je sestavljen iz dveh korakov:

1. Formiranje *eniškega komplementa* (angl. one's complement): ustvarimo ga tako, da ničle v bitnem vzorcu zamenjamo z enkami in obratno.
2. Prištevanje števila 1 k eniškemu komplementu.

Zgornji postopek si bomo ogledali na izračunu nasprotne vrednosti (pogovorno: negiranju) števila  $4_{DEC}$ , ki smo ga zapisali v 16-bitni enoti.

$$4_{DEC} = 0000\ 0000\ 0000\ 0100_{BIN}$$

$$\begin{array}{r} \text{1. korak: } 1111\ 1111\ 1111\ 1011_{BIN} \\ \text{2. korak: } \hphantom{1111\ 1111\ 1111\ } + 1_{BIN} \\ \text{rezultat: } \hphantom{1111\ 1111\ 1111\ } 1111\ 1111\ 1111\ 1100_{BIN} \end{array}$$

Pravilnost postopka lahko preverimo na dva načina:

- Negiranje negativne vrednosti mora rezultirati z enako vrednostjo pozitivnega predznaka:

$$-4_{DEC} = 1111\ 1111\ 1111\ 1100_{BIN}$$

$$\begin{array}{r} \text{1. korak: } 0000\ 0000\ 0000\ 0011_{BIN} \\ \text{2. korak: } \hphantom{0000\ 0000\ 0000\ } + 1_{BIN} \end{array}$$

rezultat:  $0000\ 0000\ 0000\ 0100_{\text{BIN}} = +4_{\text{DEC}}$

- Rezultat seštevanja števila z lastno nasprotno vrednostjo je nič:

$$\begin{array}{r}
 1111\ 1111\ 1111\ 1100_{\text{BIN}} = -4_{\text{DEC}} \\
 0000\ 0000\ 0000\ 0100_{\text{BIN}} = +4_{\text{DEC}} \\
 \hline
 \textcircled{1} \quad 0000\ 0000\ 0000\ 0000_{\text{BIN}} = 0_{\text{DEC}}
 \end{array}$$

→ prenos

Zaradi prenosa enice kot posledice seštevanja (od 3. bita naprej do MSB) pride do njenega "izpada" izven 16-bitnega obsega. Tukaj nas ne zanima njena usoda, temveč preostali biti v besedi. Lahko omenimo le to, da se pri izvajanju te operacije v mikroprocesorju ta enica "shrani" v bit prenosa (angl. carry bit; oznaka C - glej tudi pogl. 4.1.2).

### 1.2.2 Prikaz negativnih števil v različnih formatih

Čeprav smo z dvojiškim komplementom rešili problem ponazoritve negativnih števil, narekuje njegova uporaba nekatera pravila in omejitve.

Z uvajanjem zapisa negativnih števil s pomočjo dvojiškega komplementa nismo spremenili ločljivosti posamezne enote (zloga, besede ali dvojne besede), ki je bila prej celotno na voljo le pozitivnim številom (npr. 0 -  $65535_{\text{DEC}}$  za 16-bitno enoto). Posledica tega je delitev celotnega obsega na negativno in pozitivno področje.

Oglejmo si to na primeru 8-bitne enote (slika 1. 5), pri vseh višjih pa velja podobno načelo. Brez upoštevanja dvojiškega komplementa je obseg pozitivnih vrednosti:

$$0000\ 0000_{\text{BIN}} - 1111\ 1111_{\text{BIN}} \quad 00_{\text{HEX}} - \text{ff}_{\text{HEX}} \quad 0_{\text{DEC}} - 255_{\text{DEC}}.$$

Z uvajanjem dvojiškega komplementa se je celotno področje razdelilo na dva dela:

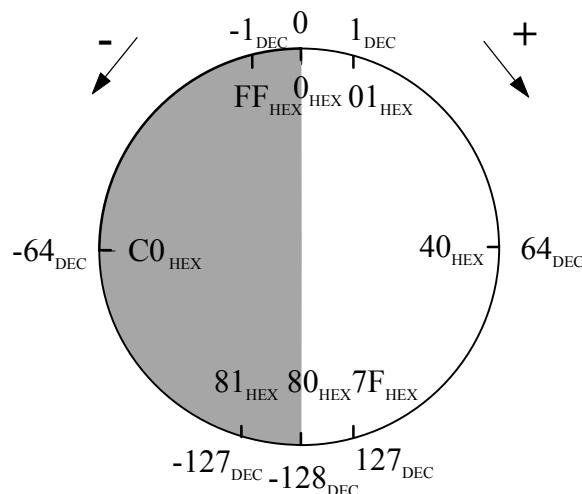
$$[-128_{\text{DEC}} \dots -1] \text{ in } [0_{\text{DEC}} \dots +127_{\text{DEC}}].$$

Kot vidimo, je po maksimalni absolutni vrednosti negativno število večje od maksimalnega pozitivnega. Razlog je v tem, da med pozitivno polovico področja ( $2^8/2 = 256/2 = 128$ ) štejemo tudi ničlo! Nasprotna vrednost števila  $-128_{\text{DEC}}$  ( $1000\ 0000_{\text{BIN}}$ ) v tej enoti ne obstaja, kar lahko dokažemo s pomočjo dvojiškega komplementa:

$$\begin{array}{l}
 \text{eniški komplement: } 0111\ 1111_{\text{BIN}} \\
 \qquad\qquad\qquad +1_{\text{BIN}} \\
 \hline
 \text{rezultat: } 1000\ 0000_{\text{BIN}} = -128_{\text{DEC}}
 \end{array}$$

Torej smo dobili enako vrednost kot pred negiranjem.

Zaradi razlike med interpretiranjem binarne kode v nepredznačenih in predznačenih (dvojiški komplement) številih bomo prve označevali s standardizirano oznako U (angl. unsigned), druge pa s S (angl. signed).



**Slika 1. 4: Prikaz pozitivnih in negativnih števil v 8-bitnem formatu**

Zelo zanimivo si je ogledati dvojiške in šestnajstiške ekvivalente nekaterih pozitivnih in negativnih števil po logiki dvojiškega ekvivalenta (slika 1. 5 in tabela 1.4). V skrajnjem desnem stolpcu so ekvivalenti binarnim zapisom v U notaciji.

| BIN       | HEX | DEC (S) | DEC (U) |
|-----------|-----|---------|---------|
| 1000 0000 | 80  | -128    | 128     |
| 1000 0001 | 81  | -127    | 129     |
| 1000 0010 | 82  | -126    | 130     |
| 1111 1110 | fe  | -2      | 254     |
| 1111 1111 | ff  | -1      | 255     |
| 0000 0000 | 00  | 0       | 0       |
| 0000 0001 | 01  | 1       | 1       |
| 0000 0010 | 02  | 2       | 2       |
| 0111 1110 | 7e  | 126     | 126     |
| 0111 1111 | 7f  | 127     | 127     |

**Tabela 1.4: Nekatere vrednosti pozitivnih in negativnih števil v 8-bitnem formatu v različnih kodah**

Takoj ugotovimo, da binarne oz. heksadecimalne kode za števila od 0 - 127<sub>DEC</sub> v S in U notaciji enako interpretiramo. Do razlik pride pri negativnih S številih. Najmanjše (po absolutni vrednosti) negativno število po S notaciji (-1<sub>DEC</sub>) je enako največjemu številu po U notaciji (255). Večanje negativnih S števil je analogno manjšanju vrednosti v U notaciji, kar nazorno prikažemo s krogom na sliki 1. 5.

Iz povedanega lahko ugotovimo nekaj dejstev, ki veljajo tudi za S števila v besednih in dvobesednih enotah:

- pri največjem pozitivnem številu je MSB enak nič, vsi ostali biti pa so enaki ena: ( $7\text{fff}_{\text{HEX}} = 32767_{\text{DEC}}$  oz.  $7\text{fff}_{\text{HEX}} = 2\ 147\ 483\ 647_{\text{DEC}}$ ),
- pri negativnem številu z največjo absolutno vrednostjo je MSB enak ena, vsi ostali biti pa so enaki nič ( $8000_{\text{HEX}} = -32768_{\text{DEC}}$  oz.  $8000\ 0000_{\text{HEX}} = -2\ 147\ 483\ 648_{\text{DEC}}$ ),
- negativno število z najmanjšo absolutno vrednostjo ( $-1_{\text{DEC}}$ ) je sestavljeno iz samih enic ( $\text{ffff}_{\text{HEX}}$  oz.  $\text{ffff}\ \text{ffff}_{\text{HEX}}$ ).

MSB je pri S notaciji vedno indikacija predznaka (1 za negativno, 0 za pozitivno število), za ugotovitev absolutne vrednosti pa moramo izračunati dvojiški komplement.

V tabeli 1.5 sta še enkrat zbrana oba zapisa v pomembnejših formatih in dveh kodah.

| Formati   | Predznačeno (S)           |                             | Nepredznačeno (U)             |                      |
|-----------|---------------------------|-----------------------------|-------------------------------|----------------------|
|           | DEC                       | HEX                         | DEC                           | HEX                  |
| Byte      | $[-128 \dots 127]$        | $[80 \dots 7F]$             | $[0 \dots 255]$               | $[0 \dots FF]$       |
| Word      | $[-32768 \dots 32767]$    | $[8000 \dots 7FFF]$         | $[0 \dots 65535]$             | $[0 \dots FFFF]$     |
| Long word | $[\pm 2,147 \times 10^9]$ | $[80000000 \dots 7FFFFFFF]$ | $[0 \dots 4,295 \times 10^9]$ | $[0 \dots FFFFFFFF]$ |

**Tabela 1.5: Povzetek U in S zapisov za različne enote**

Dvojiškemu komplementu smo namenili veliko prostora zato, ker ga pogosto srečamo pri delu s procesorji, zlasti ko programiramo v zbirniku ali ko interpretiramo vrednosti bipolarnih A/D in D/A pretvornikov. V nadaljevanju si bomo ogledali primera možnih pasti pri računanju s predznačenimi števili.

### 1.2.2.1 Presežek

*Presežek* (angl. overflow) je ena pogostih posledic izvajanja aritmetičnih operacij v procesorju in lahko pripelje do napačne interpretacije rezultatov. Presežek je posledica nezmožnosti zapisa rezultata operacije med dvema veljavnima številoma v enoti operandov. Posredno je v to "vpletena" tudi logika dvojiškega komplementa.

Vzemimo za primer seštevanje dveh 16-bitnih negativnih števil  $-30000_{\text{DEC}}$  in  $-8000_{\text{DEC}}$ . Pričakovani rezultat je  $-38000_{\text{DEC}}$ . Iz tabele 1.6 je razvidno, da tega rezultata v 16-bitni enoti ne moremo zapisati. Oglejmo si operacijo na nivoju binarnih števil (dvojiška komplementa smo že izračunali):

$$\begin{array}{r}
 1000\ 1010\ 1101\ 0000_{\text{BIN}} \\
 + 1110\ 0000\ 1100\ 0000_{\text{BIN}} \\
 \hline
 \textcircled{1} 0110\ 1011\ 1001\ 0000_{\text{BIN}}
 \end{array}
 \quad
 \begin{array}{r}
 8ad0_{\text{HEX}} \\
 + e0c0_{\text{HEX}} \\
 \hline
 6b90_{\text{HEX}}
 \end{array}
 \quad
 \begin{array}{r}
 -30000_{\text{DEC}} \\
 + (-8000_{\text{DEC}}) \\
 \hline
 27536_{\text{DEC}}
 \end{array}$$

→ prenos

Tudi brez podrobne analize rezultata izračuna lahko ugotovimo, da smo kot rezultat seštevanja dveh negativnih števil dobili pozitivno število, saj je MSB rezultata enak nič. Torej je rezultat napačen, kar lahko ugotovimo iz njegove decimalne interpretacije. Ta pa ni takšna, kot bi jo pričakovali.

Zaradi presežka se postavi poseben bit v statusnem registru (običajno označen z V ali OV), lahko pa se generira ustrezna prekinitvev (glej pogl. 4.1.2).

### 1.2.2.2 Prenos števil v višjo enoto

Pri prenosu števil iz nižje enote (npr. zloga) v višjo enoto (npr. besedo) je tudi treba upoštevati logiko dvojiškega komplementa. V bistvu gre tukaj za pravilno preslikavo MSB. Oglejmo si primer prenosa dveh 8-bitnih števil v 16-bitno enoto.

- **Pozitivno število**

Pri prenosu moramo paziti na to, da se vsi "dodani" biti na levi strani zapolnijo z ničlami (štivo v oklepajih označuje enoto zapisa):

$$\begin{array}{ccc} 75_{\text{HEX}}(8) & \rightarrow & 0075_{\text{HEX}}(16) \quad \text{al i} \\ 0111 \ 0101_{\text{BIN}}(8) & \rightarrow & 0000 \ 0000 \ 0111 \ 0101_{\text{BIN}}(16). \end{array}$$

- **Negativno število**

Pri prenosu se vsi dodani dodani biti zapolnijo z enicami:

$$\begin{array}{ccc} 85_{\text{HEX}}(8) & \rightarrow & ff85_{\text{HEX}}(16) \quad \text{al i} \\ 1111 \ 0101_{\text{BIN}}(8) & \rightarrow & 1111 \ 1111 \ 1000 \ 0101_{\text{BIN}}(16). \end{array}$$

Nekaj podobnega se dogaja tudi pri operacijah aritmetičnega pomika (angl. arithmetic shift) v levo ali desno (glej tudi pogl. 4.2.4).

Zaradi predznaka na primer pri hardverski vezavi 12-bitnega bipolarnega A/D (ali D/A) pretvornika na 16-bitno podatkovno vodilo poravnamo MSB naprave in MSB vodila (štirje odvečni skrajnje desni biti vodila pri tem ne igrajo nobene vloge; glej tudi pogl. 10.3.4.4.1).

Osnovno pravilo pri vseh opisanih operacijah je torej prenos MSB iz nižje v višjo enoto oz. njegovo ohranjanje.

### 1.3 Števila s plavajočo vejico

Aritmetika večine procesorjev, ki se uporablja v krmilnih in regulacijskih nalogah sloni na uporabi celih števil. Ta omejitev zahteva od programerja dodaten trud pri iskanju optimalnega izkoriščanja številskega obsega. Zato je veliko enostavnejše delati z *realnimi števili*.

Slednje v mikroračunalnikih najpogosteje zapisujemo v obliki števil s *plavajočo vejico* (angl. floating point) [12]. Veljavni ANSI/IEEE standard 754-1985 za aritmetiko binarnih števil s plavajočo vejico je bil sprejet leta 1985. Na tem projektu je več kot deset let delalo 92 strokovnjakov iz vrst matematikov, računalničarjev in ostalih inženirjev z univerz in iz industrije. Razlog za tako obsežno delo je bila zmeda na področju zapisa realnih števil, saj so nekateri proizvajalci uporabljali binarne ali decimalne zapise, v (bivši) Sovjetski zvezi pa celo *trinarne* (baza 3). Celo med binarnimi računalniki so obstajale inačice, ki so za bazo imele število 2, 8 ali 16.

V tem poglavju bomo opisali t.i. *format z dvojno natančnostjo* (angl. double precision format). Poleg tega obstajata še format z *enojno natančnostjo* (angl. single precision) ter format s *razširjeno natančnostjo* (angl. extended precision).

Števila s plavajočo vejico so povečinoma normirana. To pomeni, da jih lahko zapišemo v obliki

$$x = \pm(1+f) \cdot 2^e,$$

kjer *f* *mantisa* ali *ulomek* (angl. mantissa, fraction), *e* pa *eksponent*. Mantisa mora zadostiti pogoju normiranja (glej tudi poglavje 10):

$$0 \leq f < 1.$$

Mantiso zapišemo v 52-bitnem formatu. To pomeni, da mora biti  $2^{52} \cdot f$  celo število v območju

$$0 \leq 2^{52}f < 2^{53}. \quad (1.3)$$

Območje eksponenta *e* je:

$$-1022 \leq e \leq 1023. \quad (1.4)$$

Realna števila s plavajočo vejico in z dvojno natančnostjo zapisujemo v 64-bitnem formatu:

| 1        | 11       | 52       |
|----------|----------|----------|
| <i>s</i> | <i>e</i> | <i>f</i> |

Bit *s* določa predznak (angl. sign) števila, eksponent je lahko pozitiven (zelo velika števila) ali negativen (zelo majhna števila), mantisa pa nepredznačena.

Za eksponent je rezerviranih enajst bitov, torej je njegova ločljivost  $2^{11} = 2048$ . Iz podpoglavlja o dvojiškem komplementu celih števil je razvidno, da je interval (pozitivna in negativna števila) za ta format  $[-1024, +1023]$ . Zastavlja se vprašenje, zakaj dve največji negativni števili ( $-1024$  in  $-1023$ ) nista zajeta v intervalu eksponenta (1. 3)? Odgovor leži v zapisu ekstremnih primerov, ko:

- števila presegajo dovoljeno področje zapisa ali
- jih sploh ne moremo zapisati kot število.

V prvem primeru je takšen rezultat t.i. *neskončno število - Inf* (angl. infinity). Takrat je  $f = 0$  in  $e = 1024$ , pri tem pa morajo veljati tudi relacije kot so:  $1/\text{Inf} = 0$ ,  $\text{Inf} + \text{Inf} = \text{Inf}$ ....

Včasih se pa rezultata aritmetične operacije sploh ne da zapisati v obliki realnega števila. Te vrednosti označujemo kot *NaN* (angl. Not-a-Number: “ni veljavno število”) in so posledica operacij, kot so  $0/0$  ali  $\text{Inf} - \text{Inf}$ .

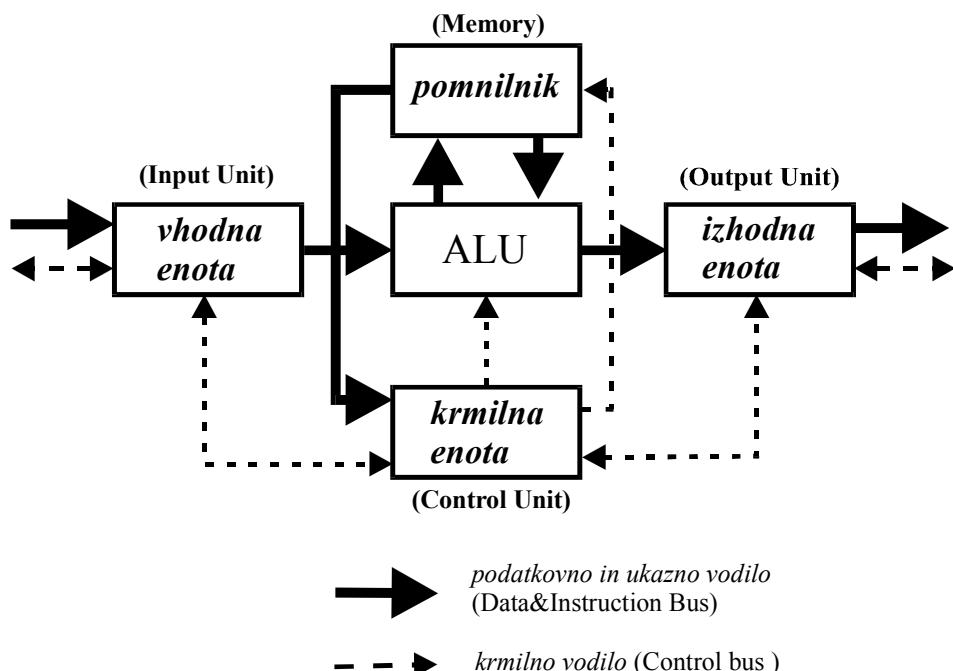
## 2. ZGRADBA MIKROPROCESORJA

### 2.1 Mikroprocesor

Računalnik si pogosto predstavljamo, kot "inteligentni" vmesnik med neobdelanimi vhodnimi podatki in krmilnimi ali informacijskimi izhodi [16]. Tej definiciji bi lahko ustrežali tudi mehanski sistemi, analogna in digitalna vezja ipd., zato mora računalnik vsebovati še:

- del, ki je sposoben izvajanja določenih aritmetičnih in logičnih operacij (ti. *aritmetično-logična enota - ALU*),
- *krmilni del*, ki je sposoben sprejemati odločitve,
- *pomnilnik*, v katerega lako shranjujemo podatke, vmesne in končne rezultate ter skupine ukazov (inštrukcij) za izvajanje želenih operacij,
- *vhodne enote* (npr. tipkovnico, A/D pretvornik itd.) za zajemanje, problikovanje in vnos podatkov iz okolice ter
- *izhodne enote* (npr. D/A pretvornik, prikazovalnik s tekočimi kristali - LCD, tiskalnik itd.) za komunikacijo s človekom, drugim računalnikom ali krmiljenje okoliških naprav.

Na ta način je zgrajen osnovni model t.i. von Neumannovega računalnika (slika 2. 1).

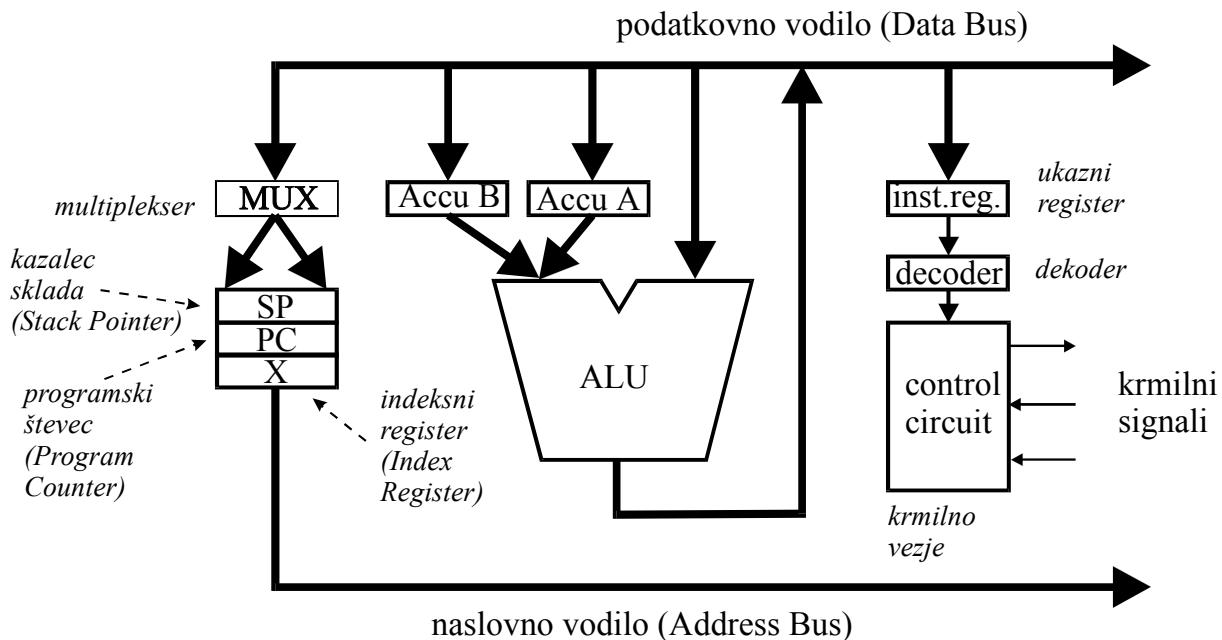


Slika 2. 2: Von Neumanov model računalnika (ALU = aritmetično-logična enota)

Z razvojem tehnologije in večanjem števila tranzistorskih elementov na enoto površine (npr. VLSI - angl. Very Large Scale Integration: več kot 100 000 tranzistorjev na čipu) se je odprla

možnost združevanja opisanih elementov na enem samem integriranem vezju - *čipu* (angl. chip).

*Mikroprocesor* (angl. microprocessor,  $\mu$ P) je v osnovi integrirano vezje, ki vsebuje ALU, osnovno krmilno enoto ter elementarni pomnilnik (*registre*, glej pogl. 4)<sup>1</sup>, torej sestavne dele tistega, kar pri večjih digitalnih računalnikih ali mikrokrmilnikih imenujemo *centralna procesna enota* (CPU, angl. Central Processor Unit). Slika 2. 3 kaže osnovne elemente enostavnega CPU ali mikroprocesorja. Ti osnovni gradniki so v taki ali drugačni obliki prisotni tudi pri večini novejših in bolj zapletenih procesorskih enotah (konkretni zgled je opisan v poglavju 4).

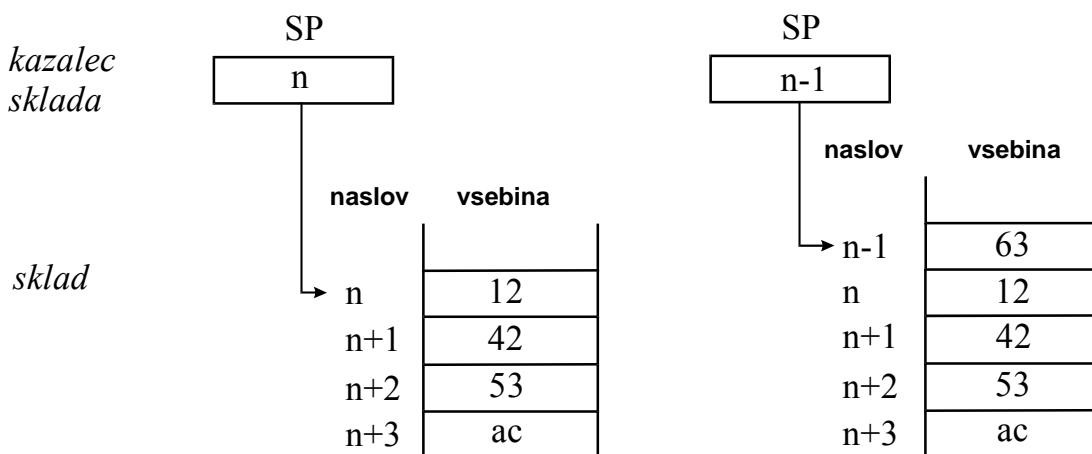


Slika 2. 4: Blokovna shema enostavnega CPU

- ALU je zadolžen za izvajanje aritmetičnih (npr. seštevanje) in logičnih operacij (npr. logična funkcija IN) nad binarnimi podatki. ALU je običajno pridružen tudi *statusni register* (angl. Status Register - SR). Njegova vsebina so biti, t.i. *zastavice* (angl. flag), ki kažejo na status rezultata aritmetične ali logične operacije (npr. rezultat manjši od ničle, bit prenosa itd.).
- Ukazi, ki se bodo izvajali v CPU, so shranjeni v nekem zunanjem pomnilniku v obliki binarnega zapisa. Binarni zapis posameznega ukaza je določen z ukazno kodo. Ukaze je pred izvajanjem najprej treba prenesti iz pomnilnika v *ukazni register*. Dekodiranje njene kode, to je ugotavljanje funkcije ukaza, opravlja *ukazni dekoder*. Posledica ukaza je generiranje ustreznih krmilnih signalov in/ali ustreznega naslova na naslovnem vodilu.
- Prenos ukazov med zunanjimi enotami (npr. pomnilnikom) in CPU ter znotraj CPU poteka po *podatkovnem vodilu* (angl. data bus). Po njem se pranašajo tudi podatki, ki jih ti ukazi uporabljajo.

<sup>1</sup> Registrji so v bistvu pomnilniške lokacije znotraj CPU s točno določenim namenom.

- Ukazi in podatki se nahajajo v pomnilniku na enoumno določenih naslovih. Vsaki kombinaciji bitov na *naslovnem vodilu* (angl. address bus) ustreza ena lokacija v pomnilniku, na kateri se nahaja ukaz ali podatek<sup>2</sup>.
- Podatke (eden ali dva podatka, npr. sumanda pri seštevanju, operand pri negiranju), ki jih uporabljajo določeni ukazi, pred izvajanjem operacije prenesemo iz pomnilnika ali drugih registrov v delovne registre ali *akumulatorje*. Procesor na sliki 2. 5 ima le dva akumulatorja, CPU32, ki bo opisan v pogl. 4, pa ima osem podobnih registrov. Ti se imenujejo *podatkovni registri* (pogl. 4).
- *Programski števec* - PC (angl. Program Counter) je register, v katerem sa nahaja naslov inštrukcije, ki se bo izvršila. Pri linearinem poteku programa se PC inkrementira v odvisnosti od dolžine ukazov. Pri pogojnih in brezpogojnih skokih se vsebina PC napolni z destinacijo skoka.
- *Indeksni register* (angl. Index Register - X) se uporablja pri posebnem načinu naslavljanja, t.i. indeksnem naslavljaju, kjer naslov izračunamo kot seštevek vsebine indeksnega registra in nekega drugega registra ali kar naslova v inštrukciji.
- Vsebina *kazalca sklada* (angl. Stack Pointer - SP) je kazalec (angl. pointer) na zadnjo veljavno lokacijo v *skladu* (angl. stack)<sup>3</sup>. Sklad je pomnilniško področje za začasno shranjevanje določenih podatkov. Za sklad je značilno, da deluje po LIFO (angl. Last-In-First-Out) principu (slika 2. 6). Najnovejši podatek, ki ga shranimo v sklad, zasede nizji naslov od prejšnjega, torej je smiselno sklad postaviti na konec RAM pomnilnika, ker na ta način ne bo posegal v področje, ki je že zasedeno z ukazi ali s podatki. V sklad se podatki vpisujejo samodejno (npr. status CPU in vsebina PC pri skoku na prekinitveni podprogram) ali s posebnimi ukazi (npr. ukaza **PUSH** - prenos stanja iz akumulatorja v sklad in **PULL** - prenos iz sklada nazaj v akumulator). Po vnosu podatka se vsebina kazalca sklada zmanjša, po branju zadnjega podatka pa se poveča<sup>4</sup>. Zaradi takšne organizacije, kjer potekata vpisovanje in branje po točno določenem vrstnem redu, moramo biti pri manipulirjanju s skladom zelo previdni. Na sliki je prikazan sklad, ki je organiziran na bytni osnovi.



Slika 2. 7: Sklad in kazalec sklada: stanje pred (levo) in po vnosu novega podatka (desno)

<sup>2</sup> Pri nekaterih procesorjih so podatki in ukazi shranjeni na ločenih pomnilniških področjih.

<sup>3</sup> PC, X in SP imenujemo včasih tudi naslovni registri. Le-te moramo razlikovati od posebne skupine naslovnih registrov pri nekaterih procesorjih (npr. A0-A7 pri CPU32), ki lahko, med ostalim, nadomestijo indeksni register (PC in SP sta ločena registra).

<sup>4</sup> Od tod tudi naziv LIFO: zadnji vhodni podatek v sklad mora biti hkrati tudi prvi, ki ga bo zapustil.

## 2.2 Mikroračunalnik

Mikroprocesor za svoje delovanje potrebuje nekatere dodatne sklope, ki skupaj z njim sestavljajo *mikroračunalnik* (angl. microcomputer, kratica -  $\mu$ C).

### 2.2.1 Ura

Operacije v mikroprocesorju se izvajajo sekvenčno (običajna koraka sta prevzem ukaza in njegovo izvajanje: angl. *fetch* in *execute* [8]). Zaradi tega jih moramo sinhronizirati na neko referenčno frekvenco dajalnika takta - *ure* (angl. clock). Zmožnost sledenja večji frekvenci pomeni hitrejše izvajanje programa<sup>5</sup>, zato je pri izbiri mikroprocesorja osnovna frekvenca eden glavnih parametrov.

Nekateri procesorji zahtevajo za generiranje takta posebno vezje, pri drugih pa je treba le priključiti kvarčni oscilator s predpisano frekvenco.

### 2.2.2 Pomnilnik

Naloga *pomnilnikov* (angl. memory) je shranjevanje podatkov, ukazov ter vmesnih in končnih rezultatov. Pomnilnike delimo na:

- pomnilnike s sekvenčnim dostopom in
- pomnilnike z neposrednim dostopom (RAM in ROM).

V prvo skupino spadajo t.i. sekundarni pomnilniki in se zaradi počasnejšega dostopa uporabljo v glavnem za trajnejše shranjevanje (npr. magnetni diski, diskete ali trakovi). Prednost nekaterih sekundarnih pomnilnikov je v tem, da niso fiksno vezani na določen računalnik, kar omogoča enostaven prenos.

Delo s pomnilniki z neposrednim dostopom je bistveno hitrejše. Vsebino bralno-pisalnega pomnilnika (*RAM* - angl. Random Access Memory) lahko neomejeno pogosto vpisujemo ali beremo. Osnovna pomanjkljivost RAM je posledica njegove izvedbe: za ohranjanje spomina je potrebno neprekinjeno napajanje, v nasprotnem primeru se njegova vsebina izbriše. Pri procesnih računalnikih, kjer obstaja možnost izpada napajanja, moramo poskrbeti za podporo (angl. backup). To naredimo bodisi s posebno baterijo ali s kondenzatorjem, ki zaradi relativno majhne porabe RAM za nekaj časa nadomesti napajanje.

Značilno za pomnilnike tipa *ROM* (angl. Read Only Memory - bralni pomnilnik) je, da njihovo vsebino lahko samo beremo, spremenljati pa je ne moremo (vsaj ne med izvajanjem programa v realnem času). Razlikujemo nekaj podvrst ROM pomnilnikov:

---

<sup>5</sup> Pri sodobnih procesorjih, ki jih srečujemo v osebnih računalnikih (npr. Pentium), je frekvenca običajno nekaj sto MHz (npr. 450 MHz), pri procesnih mikroračunalnikih pa nekaj deset MHz (npr. 16,77, 20 ali 25 pri MC68332).

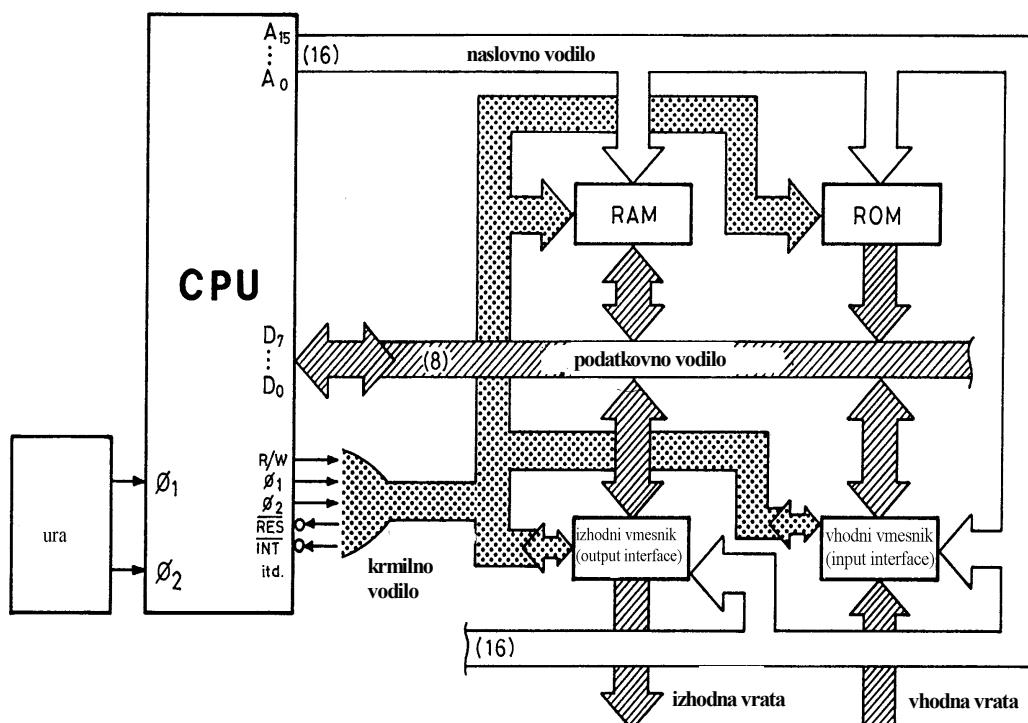
- Navadni ROM je običajno vpisan ("zapečen") že v fazi izdelave in njegove vsebine ne moremo spremeniti.
- V PROM (angl. Programmable ROM) lahko uporabnik vpisuje samo enkrat, za kar rabi posebno programirno napravo. Vsebine pozneje ni več možno spremenjati.
- EPROM (angl. Erasable PROM) je PROM, ki ga programiramo s programirno napravo, lahko ga pa tudi izbrišemo tako, da ga v posebni napravi osvetlimo z ultravijoličnimi žarki. Zato imajo EPROM čipi posebno stekleno okence, ki omogoča dostop UV žarkov do vezja. Postopek je možno ponoviti nekaj stokrat.
- EEPROM, E<sup>2</sup>ROM ali EAROM (angl. Electrically EPROM) je podoben EPROM-u. Razlika je v tem, da njegovo vsebino brišemo in vpisujemo s posebnim električnim vezjem. Zaradi relativne dolgotrajnosti postopka in potrebe po dodatnem vezju se te metode ne moremo poslužiti med izvajanjem uporabniškega programa v mikroračunalniku (on-line). Posebni t.i. Flash inačici EEPROM vezij je posvečeno posebno poglavje.

Programiranje vseh programljivih ROM pomnilnikov je možno le v posebnih programirnih napravah. Zato te pomnilnike pri izdelavi mikroračunalniškega vezja običajno postavimo na posebna podnožja, ki omogočajo prenosljivost čipa.

### 2.2.3 Vhodno-izhodni vmesniki

*Vmesnik* (angl. interface) je posrednik med mikroračunalnikom in okolico. Največkrat gre za časovno uskladitev in prilagoditev signalnih nivojev. Signali iz procesa so lahko v binarni (npr. končno stikalo) ali analogni obliku (npr. meritni člen za ugotavljanje nivoja tekočine). Enako velja tudi za izhodna povelja (npr. digitalni - vklapljanje ventila; analogni - želeni statorski tok motorja).

Slika 2. 8 kaže primer mikroračunalnika z do sedaj opisanimi sestavnimi deli.



### **Slika 2. 9: Osnovna zgradba mikroračunalnika**

Pri aplikacijah v realnem času pogostokrat rabimo posebno izvedbe mikroračunalnikov. Njihova izdelava je zelo zamudno delo, ki zahteva veliko specialističnega znanja. Tudi cena takega, po "meri" narejenega mikroračunalnika, ni zanemarljiva. Poleg tega je zelo težko doseči majhne gabarite naprave, kar posledično vpliva tudi na hitrost in zanesljivost delovanja. Daljše povezave med integriranimi vezji so namreč bistveno bolj dovetne za motnje<sup>6</sup>.

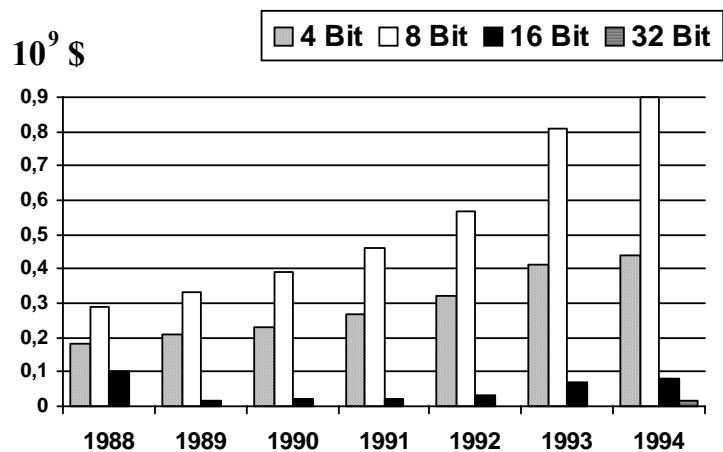
Napredki na področju minimizacije mikroelektronskih komponent so sčasoma pripeljali do integracije nekaterih opisanih sestavnih delov na enem čipu. Tako smo dobili t.i. "računalnik na čipu" ali *mikrokrmilnik* (angl. microcontroller), med katere sodi tudi Motorolin MC68332, ki ga bomo podrobno opisali v nadaljevanju tega učbenika. Večina mikrokrmilnikov vsebuje poleg CPU še pomnilnika RAM in ROM (običajno PROM ali EPROM), digitalne vhode in izhode, enoto za serijsko komunikacijo, programirljive števce in časovnike itd. Očitna prednost mikrokrmilnikov se kaže zlasti v zmanjšanju števila zunanjih elementov, običajno je pa le potrebna nadgradnja v skladu z zahtevami procesa. Dodatne komponente na mikrokrmilniku vplivajo na njegovo ceno. Zato ponujajo proizvajalci mikrokrmilnike, ki so namenjeni za posebne naloge. Tako je npr. mikrokrmilnik TMS320F240 predviden za regulacijo električnih motorjev. Zaradi specifične namembnosti ga lahko uvrstimo med vezja za specifične aplikacije (angl. ASIC - Application Specific Integrated Circuit).

## **2.3 Tržni delež posameznih tipov mikroprocesorjev**

V tem učbeniku bomo govorili o 8-, 16- in 32-bitnih mikroprocesorjih, z osnovno frekvenco ure okrog nekaj deset MHz, torej o procesorjih, ki se v primerjavi z najbolj znanimi iz osebnih računalnikov zdijo zastareli. Dejstvo je, da so takšni procesorji najbolj zastopani na trgu (slika 2. 10). Razlogov za to je veliko. Med najpomembnejše sodi zahtevnost krmilne ali regulacijske naloge. Povečinoma so takšni procesorji relativno nezahtevni z ozirom na hitrost in obseg funkcij in jih omenjeni procesorji uspešno rešujejo (npr. krmiljenje elektronike v avtomobilih: vžig, krmiljenje motorjev za brisalnike, dvig stekel itd., krmiljenje in regulacija elektromotorskih pogonov, aplikacije v telekomunikacijah, alarmne naprave, inteligentni polnilniki akumulatorskih baterij itd.). Tovrstni procesorji so se izkazali kot zanesljivi in jih sprembla celotna aparatura in programska oprema ter veliko pomožnih gradiv (literatura, vključno s brezplačnimi in enostavnimi dosegljivimi primeri uporabe, ki jih ponujajo sami proizvajalci). Neposredna posledica tega je tudi masovna proizvodnja in s tem nižja cena komponent (npr. maloprodajna cena že precej starega, še vedno pa zelo zmogljivega 16-bitnega procesorja MC68000, je manj kot 5\$). Izkušnje projektanta se kažejo tudi pri izboru ustreznega procesorja: "predober" procesor je običajno precej dražji. Temu moramo dodati še določeno konzervativnost uporabnikov, ki se nerada odločajo za modne novitete; deluječe procesorje zamenjamo z zmogljivejšimi le takrat, ko se povečajo zahteve procesa.

---

<sup>6</sup> Zelo značilen primer za to so podatki proizvajalcev o številu taktov potrebnih za komunikacijo s pomnilnikom, ki se nahaja na samam čipu, in z zunanjim pomnilnikom. Na splošno velja, da zahteva komunikacija z zunanjim pomnilnikom več časa.



*Slika 2. 11: Tržni delež mikroprocesorjev z ozirom na dolžino osnovne besede*

## 3. MIKROKRMLNIK MC68332

V tem učbeniku si bomo podrobneje ogledali dva mikrokrmlnika za aplikacije v močnostni elektroniki oz. elektromotorskih pogonih. O značilnostih digitalnega signalnega procesorja TMS320F240 bomo več govorili v pogl. 11, za referenčni mikrokrmlnik, skozi katerega bomo predstavili vse splošno zanimive arhitekturne značilnosti, pa smo izbrali MC68332 firme Motorola.

### 3.1 Kratka zgodovina nastanka MC68332

#### 3.1.1 Procesor MC68000

Za nas zanimivo obdobje se je začelo sredi sedemdesetih let. Takrat je Motorola predstavila svoj zelo znani 8-bitni mikroprocesor MC6800. Nedolgo za tem je sledila še dodatna periferija, s čimer je celotna družina postala standard na področju procesne tehnike. Pet let pozneje je nastal 16-bitni MC68000 s pripadajočim okoljem.

Popularnost teh mikroprocesorjev je posledica preproste arhitekture in nezahtevne povezav z zunanjimi elementi. Oba procesorja sta bila osnova za nastanek dveh različnih družin mikroprocesorjev in mikrokrmlnikov, kot sta na primer 8-bitni MC68HC11 (osnova MC6800) in 32-bitni MC68332 (osnova MC68000).

Lastnosti originalnega MC68000 so:

- CISC (Complex Instruction Set Computer) - računalnik s kompleksnim naborom ukazov,
- osem 32-bitnih podatkovnih registrov za splošno rabo (D0-D7),
- osem 32-bitnih naslovnih registrov za splošno rabo (A0-A7); A7 je kazalec sklada ("stack pointer"),
- 32-bitni programski števec (Program Counter - PC) - linearno 4 Gbyte - brez segmentov in strani,
- 16-bitno zunanje podatkovno vodilo, ki je lahko povezano z 8- ali 16-bitnim pomnilnikom in periferijo,
- 16 Mbytov linearne naslovljivega prostora (24 bit),
- 56 tipov ukazov (z različnimi načini naslavljanja več kot 1000 ukazov),
- "memory mapped" I/O - vhodi/izhodi so naslovljeni kot pomnilniški prostor,
- 14 načinov naslavljanja,
- 5 glavnih tipov podatkov (bit, byte, BCD, beseda in dolga beseda),
- 7 nivojev prekinitev,
- 5 V NMOS dinamična struktura.

Po naslednikih osnovne verzije MC68008 in MC68010 se je pojavil procesor MC68020, ki je nekaj let za tem postal osnovni sestavni del mikrokrmlnika MC68332. MC68020 že vsebuje 32-bitno naslovno in 32-bitno podatkovno vodilo ter možnost naslavljanja "cikel po cikel" 8-, 16- in 32-bitne periferije. Procesor je združljiv s svojimi predhodniki, dodanih pa je še 20

novih ukazov ter 256 bytni ukazni predpomnilnik. MC68020 so sledili novejši procesorji MC68030, MC68040 MC68060.

Naslednja tabela kaže primerjavo med posameznimi procesorji družine 68XXX.

|                                 | <b>68000</b>    | <b>68010<sup>I</sup></b> | <b>68020</b>                   | <b>68030</b> | <b>68040</b> | <b>68060</b> |
|---------------------------------|-----------------|--------------------------|--------------------------------|--------------|--------------|--------------|
| <b>Podatkovno vodilo</b>        | 16              | 16                       | 8/16/32                        | 8/16/32      | 32           | 32           |
| <b>Naslovno vodilo</b>          | 24              | 24                       | 32                             | 32           | 32           | 32           |
| <b>Virtualni pomnilnik</b>      | -               | DA                       | DA                             | DA           | DA           | DA           |
| <b>Ukazni predpomnilnik</b>     | -               | 3                        | 256                            | 256          | 4096         | 8192         |
| <b>Podat. predpomnilnik</b>     | -               | -                        | -                              | 256          | 4096         | 8192         |
| <b>Upravnik spomina</b>         | 68541/<br>68851 | 68541/<br>68851          | 68851                          | DA           | DA           | DA           |
| <b>FPU<sup>II</sup> vmesnik</b> | -               | -                        | 68881/<br>68882 <sup>III</sup> | interno      | interno      | interno      |
| <b>Vgrajena FPU</b>             | -               | -                        | -                              | -            | DA           | DA           |
| <b>Tip podat cikla</b>          | asnahr.         | asnahr.                  | asnahr.                        | asnahr.      | oba          | sinhr.       |
| <b>Sprem. dolžine vodila</b>    | -               | -                        | DA                             | DA           | 68150        | 68150        |

#### Pripombe:

- I. 68010, 68008 in 68541 se pri Motoroli ne proizvajajo več (MC68008 je MC68000 z 8-bitnim zunanjim podatkovnim vodilom).
- II. FPU (angl. "Floating Point Unit"): enota za operacije s plavajočo vejico (matematični koprocesor), vsebuje šest 80-bitnih registrov.
- III. MC68882 je izboljšana verzija MC68881.

#### 3.1.2 Mikrokrmlniki serije MC68300

Iz procesorjev družine MC68XXX so se razvili mikrokrmlniki serije MC683XX, med katere sodi tudi MC68332. Serija sloni na procesorju CPU32, ki je zasnovan na MC68020. Naslednja tabela podaja karakteristike mikrokrmlnikov serije MC683XX.

|                        | <b>68330</b> | <b>68331</b>     | <b>68332</b> | <b>68F333</b> | <b>68340</b> | <b>68341</b> |
|------------------------|--------------|------------------|--------------|---------------|--------------|--------------|
| <b>Jedro CPU</b>       | CPU32        | CPU32            | CPU32        | CPU32         | CPU32        | CPU32        |
| <b>TPU (časovnik)</b>  | -            | -                | DA           | DA            | -            | -            |
| <b>DUART</b>           | -            | -                | -            | -             | DA           | -            |
| <b>Statični RAM</b>    | -            | -                | 2 Kbyte      | 4 Kbyte       | -            | -            |
| <b>Flash EEPROM</b>    | -            | -                | -            | 64 Kbyte      | -            | -            |
| <b>A/D pretvornik</b>  | -            | -                | -            | DA            | -            | -            |
| <b>Serijska vrata</b>  | 2            | 2                | 2            | 2             | -            | 2            |
| <b>DMA</b>             | -            | DA               | -            | DA            | 2            | DA           |
| <b>Časovnik</b>        | 2            | GPT <sup>I</sup> | -            | 1             | 2            | DA           |
| <b>Paralelna vrata</b> | 2            | -                | -            | 18 bit        | 2            | 2            |
| <b>Chip select</b>     | 4            | 12               | 12           | 9             | 4            | 8            |

I. GPT (General Purpose Timer): časovnik za splošne namene.

## 3.2 Značilnosti mikrokrmlnika MC68332

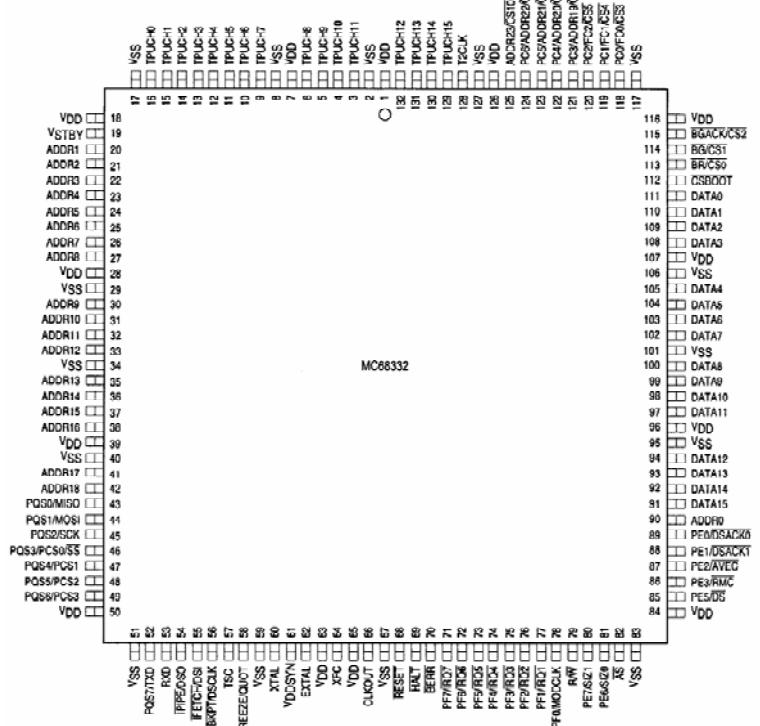
MC68332 se še pogosto uporablja v sodobnih krmilnih in regulacijskih sistemih, čeprav ne sodi med najsodobnejše mikrokrmlilnike. Njegove poglavite značilnosti so posrečena kombinacija podsestavov, dobra podpora ter enostavnost in zanesljivost. Za nas bo zanimiv predvsem z didaktičnega stališča, saj združuje arhitekturo klasičnih procesorjev in splošno uporabne module (npr. za serijsko komunikacijo, digitalne vhodno/izhodne enote, zajemanje ali generiranje časovno odvisnih pulzov itd.).

Glavne značilnosti MC68332 so:

- CISC (Complex Instruction Set Computer) - računalnik s kompleksnim naborom ukazov.
- Osem 32-bitnih podatkovnih registrov za splošno rabo (D0-D7).
- Osem 32-bitnih naslovnih registrov za splošno rabo (A0-A7); A7 je kazalec sklada (“stack pointer”).
- 32-bitni programske števec (Program Counter - PC) - linearno 4 Gbyte - brez segmentov in strani.
- 16-bitno zunanje podatkovno vodilo, ki je lahko povezano z 8- ali 16-bitnim pomnilnikom in periferijo.
- 16 Mbytov linearne naslovne prostora (24-bitno vodilo).
- Dvoprocesorski sistem (CPU in TPU).
- Modularna zgradba.
- Frekvenca 16,78 MHz, 20,97 MHz ali 25 MHz, enotno napajanje 5 V ali 3 V (odvisno od inačice)<sup>1</sup>.
- HCMOS (High Density Complementary Metal-Oxyde Semiconductor) tehnologija - 422000 tranzistorjev.
- “Memory mapped” I/O - vhodi/izhodi so naslovljeni kot spomin.
- 100 ukazov.
- 14 načinov naslavljanja.
- 5 glavnih tipov podatkov (bit, byte, BCD, beseda in dolga beseda).
- 7 nivojev prekinitve.
- Majhna poraba (največ 600 mW). Računalnik lahko tudi postavimo v posebno stanje minimalne porabe (500 µW) z ukazom LPSTOP. V tem stanju ne izvaja nobenih operacij, z normalnim delovanjem pa začne ob zunanji zahtevi.
- PQFP (Plastic Quad Flat Package) ohišje (slika 3.1) s 132 nožicami.
- Kompatibilen z mikroprocesorsko družino MC68000 (od MC68010 navzgor).
- Maksimalno število vhodno/izhodnih diskretnih nožic: 32.
- Možnost uporabe 12 softversko nastavljenih CS signalov za naslavljanje zunanjih čipov.
- Inteligentni 16-bitni časovnik in 16 neodvisnih kanalov s programirljivimi funkcijami zajemanja ali generiranja pulzov (TPU).

---

<sup>1</sup> V našem primeru bomo obravnavali inačico z urino frekvenco 16,77 MHz in napajalno napetostjo 5 V.



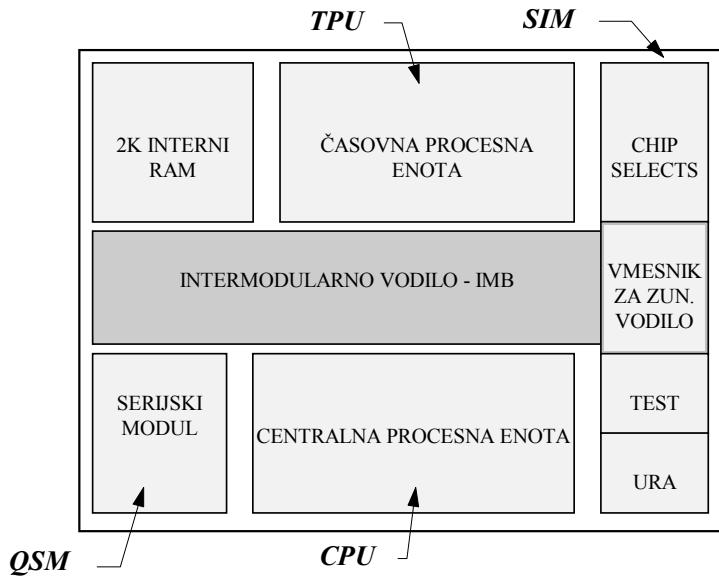
*Slika 3.1: Izgled mikrokrumilnika MC68332*

Mikrokrumilnik sestavlja naslednje funkcijeske enote - moduli (slika 3.2):

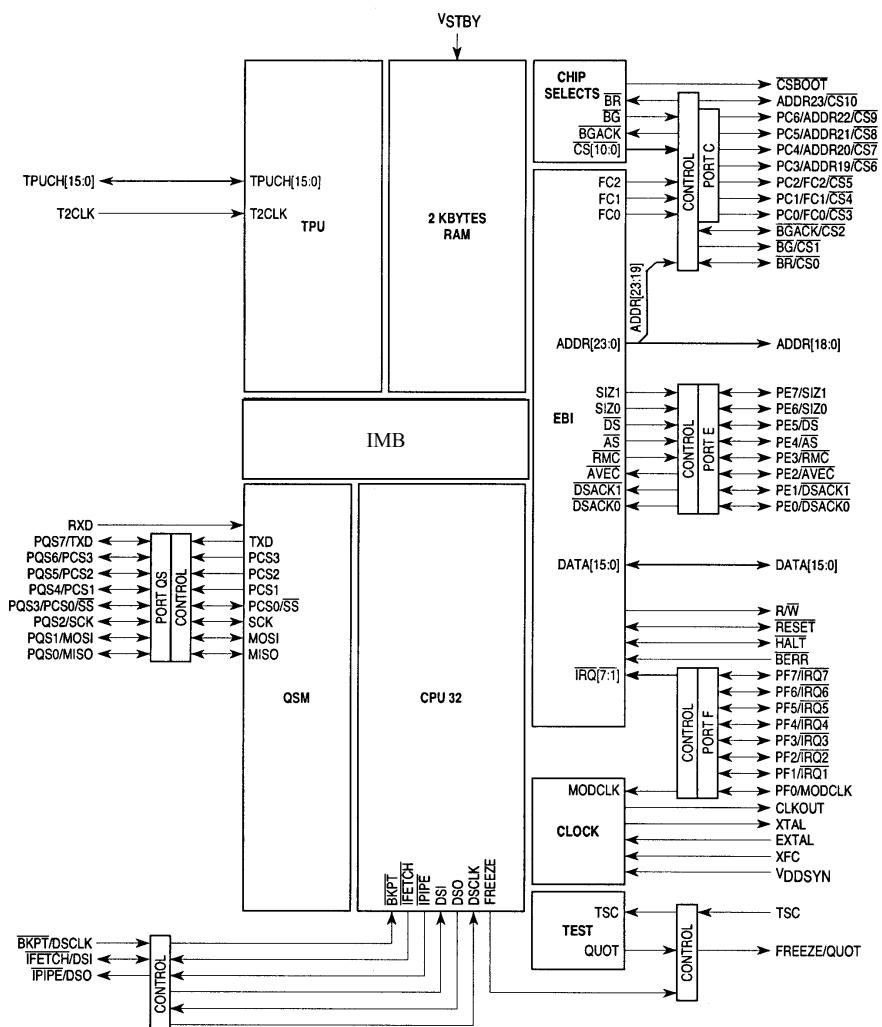
- **Centralna procesna enota** (Central Processor Unit) - CPU: glavni del krmilnika, ki nadzira delovanje ostalih modulov.
  - **Časovna procesna enota** (Time Processor Unit) - TPU je drugi procesor, ki je zadolžena za časovne funkcije.
  - **Serijski modul** (Queued Serial Module) - QSM je zadolžen za asinhronsko (SCI) in sinhronsko (QSPI) serijsko komunikacijo s periferijo.
  - Interni 2 Kbyte **RAM** (Standby RAM)<sup>2</sup> omogoča najhitrejše shranjevanje spremenljivk.
  - **Modul za izbiro čipov** (Chip Selects) omogoča minimiziranje hardverskega dela za naslavljjanje periferije. Skupaj z naslednjimi tremi podmoduli sestavlja Modul za integracijo sistema (System Integration Module - SIM).
  - Notranja **sistemska ura** - (System Clock) določa takt izvajanja operacij.
  - **Testni modul** (Test Module) je namenjen za tovarniško testiranje pravilnosti delovanja.
  - **Vmesnik k zunanjem vodilu** (External Bus Interface) - EBI povezuje krmilnik z okolico preko vodil ali digitalnih vhodno/izhodnih nožic.
  - **Medmodularno vodilo** (Intermodular Bus) - IMB povezuje ostale module.

Slika 3.3 kaže funkcije nožic MC68332.

<sup>2</sup> Ob izpadu napajanja lahko vsebino RAM-a na čipu ohranimo s pomočjo zunanjega vira (baterije) na nožici  $V_{STBY}$  (min. 3,0 V).



Slika 3.2: Principiellna blokovna shema mikroračunalnika MC68332



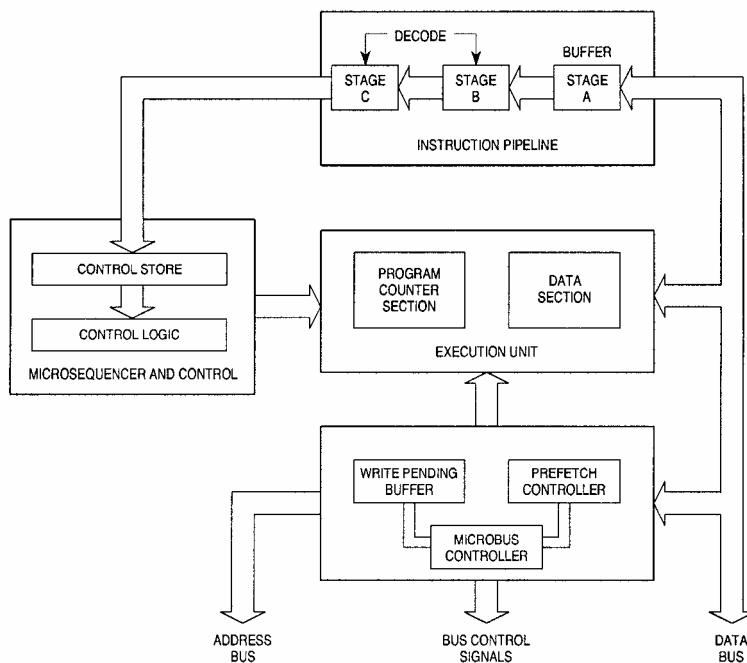
Slika 3.3: Označe nožic mikrokrumilnika MC68332

## 4. CENTRALNA PROCESNA ENOTA CPU32

### 4.1 Zgradba centralne procesne enote CPU32 mikrokrmlnika MC68332

V tem delu bomo opisali arhitekturo in nabor ukazov 32-bitne centralne procesne enote mikrokrmlnika MC68332. Ta je v mnogočem podobna ostalim CPU, ki so prikazani v poglavju 11.

Centralna procesna enota CPU32 je nastala iz mikroprocesorja MC68020 in je osnovni del vseh mikrokrmlnikov iz serije MC683xx. Njena naloga je izvajanje uporabniškega programa ter koordiniranje delovanja ostalih podmodulov. Navzdol je združljiva z družino procesorjev MC68000. CPU32 lahko programiramo v višjih programskeh jezikih HLL (High Level Languages), omogoča različne načine naslavljanja, poleg tega pa vsebuje še nekaj novih ukazov v primerjavi z MC68020. To so predvsem izboljšane (hitrejše) funkcije množenja, deljenja in pomika. Podatkovni registri podpirajo operacije z 8-, 16- ali 32-bitnimi operandi. V tem poglavju se ne bomo ukvarjali z interno arhitekturo CPU (slika 4. 1), saj nas s stališča uporabnika zanimajo le segmenti, ki so nam dostopni pri pisanju programov.



*Slika 4. 1: Blokovni diagram CPU32*

Za posredovanje informacij med uporabniškim programom in CPU so zadolženi *CPU registri*. Registri so del reduciranega področja hitrega RAM pomnilnika v CPU. Za razliko od navadnega RAM imajo centralne procesne enote le nekaj registrov (sodobnejši običajno 32, nekateri, npr. SPARC, celo 144). Zaradi tega je del kode, ki je potrebna za naslavljanje registrov, sestavljen samo iz nekaj bitov, za naslavljanje pomnilnika pa jih rabimo bistveno več, npr. 24. Čas dostopa do registrov je bistveno hitrejši od časa dostopa do pomnilnika. V

CPU imajo registri imena<sup>1</sup>, ki jih uporabljamo za naslavljjanje pri programiranju (npr. D0, A7, SP itd.).

CPU32 vsebuje dva skupka internih registrov:

- **registri uporabniškega modela** - (User Programming Model)
- **registri nadzornega modela** - (Supervisor Programming Model)

#### **4.1.1 Registri v uporabniškem modelu CPU**

Pri običajnem programiranju praviloma uporabljamo le prvo skupino registrov. Uporabniški registri so (slika 4. 2):

1. **Podatkovni registri** (D0 - D7),
2. **Naslovni registri** (A0 - A7),
3. **Kazalec sklada** (A7 ali USP),
4. **Programski števec** (PC),
5. **Register pogojnih kod** (Condition Code Register) - CCR.

| 31 | 23 | 15 | 7 | 0 |          |
|----|----|----|---|---|----------|
|    |    |    |   |   | D0       |
|    |    |    |   |   | D1       |
|    |    |    |   |   | D2       |
|    |    |    |   |   | D3       |
|    |    |    |   |   | D4       |
|    |    |    |   |   | D5       |
|    |    |    |   |   | D6       |
|    |    |    |   |   | D7       |
|    |    |    |   |   |          |
|    |    |    |   |   | A0       |
|    |    |    |   |   | A1       |
|    |    |    |   |   | A2       |
|    |    |    |   |   | A3       |
|    |    |    |   |   | A4       |
|    |    |    |   |   | A5       |
|    |    |    |   |   | A6       |
|    |    |    |   |   | A7 (USP) |
|    |    |    |   |   |          |
|    |    |    |   | 0 | PC       |
|    |    |    |   |   | CCR      |

**Slika 4. 2: Uporabniški registri CPU32**

<sup>1</sup> Registrji so praviloma ločeni od pomnilnika, pri nekaterih procesorjih (npr. Texas TMS99XX) pa so locirani v RAM pomnilniku. V MC68332 so nekatere lokacije v RAM pomnilniku vnaprej rezervirane za določanje načina obratovanja in parametriranje podmodulov (npr. MCR je "psevdoregister" za splošno določanje delovanja SIM modula in se nahaja na pomnilniški lokaciji fffa00<sub>HEX</sub> ali 7ffa00<sub>HEX</sub>). Takšne registre naslavljamo kot navadne pomnilniške lokacije.

#### 4.1.1.1 Podatkovni registri

*Podatkovni registri (Data Registers)* - DR se uporablja za delo s podatki (npr. za shranjevanje enega ali obeh operandov pri aritmetičnih ali logičnih operacijah ter rezultata te operacije). Naslavljamo jih s simbolom Dx, kjer je x zaporedna številka registra (0-7). Skupaj imamo na voljo osem 32-bitnih registrov. Zato CPU32 štejemo med 32-bitne procesorje. Podatkovni registri so analogni akumulatorjem pri nekaterih drugih CPU (npr. MC68HC11 vsebuje 8-bitna akumulatorja A in B, glej pogl. 11).

Večje število podatkovnih registrov (ali akumulatorjev) je zaželeno zaradi hitrejšega izvajanja programa, ker omogoča shranjevanje vmesnih rezultatov operacij kar v CPU. Na ta način se izognemo shranjevanju in ponovnemu branju podatkov v/iz zunanjega pomnilnika. Delo s pomnilnikom je praviloma počasnejše med ostalim tudi zato, ker ima zunanje podatkovno vodilo običajno manj podatkovnih linij od internega vodila v CPU. Npr. pri MC68332 imamo opravka s 16-bitnim zunanjim in 32-bitnim notranjim vodilom.

#### 4.1.1.2 Naslovni registri

32-bitne *naslovne registre (Address Registers)* - AR uporabljam za naslavljanje znotraj dovoljenega pomnilniškega področja. O njihovi funkciji bomo več govorili v poglavju o načinu naslavljanja (4.3).

#### 4.1.1.3 Kazalec sklada

*Uporabniški kazalec sklada (angl. User Stack Pointer)* - SP ali USP je 32-bitni register, njegova vsebina pa je naslov začetka (vrh) uporabniškega *sklada* (angl. User Stack). Način dela s skladom opisuje poglavje 4.2.7.2.

Kazalec sklada je pri večini procesorjev samostojni register, v CPU32 pa je to kar naslovni register A7. Kadar ni v funkciji naslavljanja sklada je enakovreden ostalim naslovnim registrom. Zaradi te dvojne narave moramo biti pri pisanju programov zelo pozorni, da področje sklada ne poseže v del pomnilnika, kjer so shranjeni podatki in program.

#### 4.1.1.4 Programska števec

*Programski števec (angl. Program Counter)* - PC je 32-bitni register, ki kaže na naslov naslednje ukaza za izvajanje. Dolžina ukaza je odvisna od tipa (dolžine) in števila operandov ter načinov naslavljanja. Zato mora pri linearinem poteku programa *dekoder inštrukcij* CPU skrbiti za ustrezno povečevanje vsebine PC. Pri skokih znotraj programa oz. na podprograme moramo vsebino PC spremeniti tako, da kaže na izbrano skočno lokacijo, kar dosežemo s posebnimi ukazi (glej tudi pogl. 4.2.7).

#### 4.1.1.5 Register pogojnih kod

Ta register je v bistvu spodnja polovica statusnega registra (glej naslednje podpoglavlje).

### 4.1.2 Registri v nadzornem modelu CPU (Supervisor Programming Model)

Nadzorne registre pri normalnem režimu izvajanja programa lahko le beremo. Spreminjamo jih lahko le v nadzornem režimu obratovanja (ob setirani zastavici S v statusnem registru). Ti registri so:

- nadzorni kazalec sklada,
- alternativna funkcijnska kodna regista,
- register baze vektorjev,
- statusni register.

V tem poglavju bomo opisali le funkciji zadnjih dveh.

*Register baze vektorjev (Vector Base Register) - VBR* je register, katerega vsebina določa začetni naslov pomnilniškega prostora, v katerem se nahajajo prekinitveni vektorji (več o tem v poglavju 4.2.8.1).

*Statusni register (Status Register) - SR* je edini 16-bitni register. Njegova funkcija je spremljanje rezultata operacij v procesorju in postavljanje temu ustreznih posebnih bitov. Zgornjih osem bitov je rezerviranih za t.i. sistemski byte, spodnjih osem bitov pa zaseda register pogojnih kod (Condition Code Register) - CCR.

| sistemske byte |          |         |         |         |          |         |         |        |        |        |        | CCR    |        |        |        |  |  |  |  | SR |
|----------------|----------|---------|---------|---------|----------|---------|---------|--------|--------|--------|--------|--------|--------|--------|--------|--|--|--|--|----|
| T1<br>15       | T0<br>14 | S<br>13 | 0<br>12 | 0<br>11 | I2<br>10 | I1<br>9 | I0<br>8 | 0<br>7 | 0<br>6 | 0<br>5 | X<br>4 | N<br>3 | Z<br>2 | V<br>1 | C<br>0 |  |  |  |  |    |

Oznake pomenijo:

|            |     |  |
|------------|-----|--|
| T1, T0     | ... | sledenje (angl. trace) programa omogočeno/onemogočeno              |
| S          | ... | nadzorni ("1") / uporabniški ("0") režim                           |
| I2, I1, I0 | ... | prekinitvena maska (nivoji dopustnih prekinitev - pogl. 4.2.8.1.1) |
| X          | ... | razširitveni (angl. extend) bit                                    |
| N          | ... | rezultat negativen   |
| Z          | ... | rezultat nič   |
| V          | ... | bit presežka (angl. overflow)                                      |
| C          | ... | bit prenosa (angl. carry).   |

Stanje bitov X, N, Z, V, C je odvisno od rezultata večine ukazov pri vseh mikroprocesorjih, zato si podrobneje oglejmo njihov pomen.

Bit Z se načeloma postavi, če je rezultat operacije med preznačenimi števili enak nič, če pa je manjši on nič, se postavi bit N. Razširitveni bit X je, podobno kot bit C<sup>2</sup>, pomemben pri seštevanju operandov, katerih format je večji od osnovne 32-bitne besede<sup>3</sup>. Oba bita sta zelo pomembna tudi pri operacijah pomika (angl. shift). Bit presežka (V) se postavi, ko je rezultat

<sup>2</sup> Pri nekaterih procesorjih imamo le enega.

<sup>3</sup> Če hočemo npr. z 32-bitnimi registri zapisati in seštetiti dve 64-bitni števili, moramo pri seštevanju zgornjih 32 bitov upoštevati morebiten bit prenosa kot posledico seštevanja spodnjih 32 bitov. Primer za to je ukaz ADDX.

operacije nemogoče izpisati v obsegu operandov. Pri seštevanju (ukaz ADD) je npr. bit V posledica operacije

$$V = Sm \bullet Dm \bullet Rm + Sm \bullet \underline{Dm} \bullet Rm,$$

kjer so Sm, Dm in Rm najbolj pomembni (MSB) biti prvega in drugega operanda ter rezultata<sup>4</sup>. Vidimo, da se bit setira, kadar sta MSB operandov enaka, MSB rezultata pa ima nasprotno vrednost. Pri seštevanju je to indikacija napake, saj pri logiki dvojiškega komplementa seštevek dveh negativnih števil ( $Sm = Dm = 1$ ) ne more biti pozitiven ( $Rm = 0$ ). Tak rezultat je očitno nemogoče zapisati v formatu obeh operandov.

V splošnem velja, da moramo pravila za setiranje omenjenih statusnih bitov preučiti za vsak ukaz posebej!

## 4.2 Nabor ukazov za CPU32

V tem poglavju si bomo ogledali nabor ukazov za CPU32. Čeprav ima vsak CPU svoj lastni nabor ukazov, je velika večina ukazov skupna vsem procesorjem. Sintakse so si tudi zelo podobne, kar omogoča relativno hiter prehod z enega na drug procesor.

V splošnem je ukaz sestavljen iz

- neobvezne *simbolične označke naslova ukaza* (npr. `lok`),
- *operacije* (npr. ADD),
- *tipa oz. širine operandov* (B - byte, W - beseda, L - dvojna beseda),
- *prvega operanda* (npr. D0),
- *drugega operanda* (npr. D1) - pri večini ukazov vsebuje tudi rezultat operacije,
- *komentarja* (za zvezdico \*):

```
lok ADD.L D0,D1      *seštevanje dveh 32-bitnih števil (long)
                  iz
                  *D0 in D1; rezultat se nahaja v D1 -
                  *prejšnja vsebina D1 je izgubljena
```

Poleg teh obstajajo tudi ukazi z enim operandom (npr. CLR.L D0) in ukazi brez operanda (npr. NOP). Sintaksa ukazov, ki so opisani v tem poglavju, velja za najnižji nivo vpisa ukazov, to je v *zbirniku* (angl. *assembler*).

V naslednjih podpoglavljih so ukazi glede na njihovo funkcijo predstavljeni v skupinah :

- **ukazi za prenos podatkov**,
- **ukazi za celoštevilčno (integer) aritmetiko**,
- **logične operacije**,
- **operacije pomika in rotacije**,
- **operacije za manipulacije z biti**,

---

<sup>4</sup> Operator “•” pomeni logično konjunkcijo (logična funkcija IN - angl. AND), “+” pa logično disjunkcijo (funkcija ALI - angl. OR). V nadaljevanju bomo negacijo bita, celotnega tipa ali signala označevali s podčrtajem (npr. CS).

- **operacije med BCD števili,**
- **ukazi za nadzor nad potekom izvajanja programa,**
- **ukazi za sistemski nadzor.**

V tabelah se uporabljo številne okrajšave, ki ponazarjajo načine naslavljanja ter funkcije posameznih ukazov:

|                    |   |
|--------------------|---|
| data               | podatek, ki je vsebovan v samem ukazu   |
| Dest               | (angl. destination) operand in končni rezultat (cilj)   |
| Source             | operand (začetna lokacija) <sup>5</sup>   |
| vektor             | lokacija prekinitvenega vektorja (glej podpogl. 4.2.8.1.1)  |
| An                 | katerikoli naslovni register (A0 - A7)  |
| Ax, Ay             | naslovna regista, ki ju uporabljamo v ukazu   |
| Dn                 | katerikoli podatkovni register (D0 - D7)  |
| Rc                 | nadzorni register (VBR, SFC, DFC)   |
| Rn                 | katerikoli podatkovni ali naslovni register   |
| Dh, Dl             | podatkovna regista, ki vsebuje višjo in nižjo polovico 64-bitnega rezultata množenja  |
| Dr, Dq<br>deljenju | podatkovna regista, ki vsebuje ostanek in količnik pri celoštevilčnem deljenju  |
| Dx, Dy             | podatkovna regista, ki ju uporabljamo pri izračunu v ukazu  |
| Dym, Dyn           | podatkovna regista, vrednosti za tabelarično interpolacijo  |
| Xn                 | indeksni register   |
| [An]               | podaljšek naslova   |
| cc                 | pogojna koda(pri pogojnih operacijah)   |
| d#                 | odmak pri naslavljaju (angl. displacement), npr. d <sub>16</sub> je 16-bitni odmak  |
| <ea>               | efektivni naslov (načeloma pomeni poljubni način naslavljanja; za natančnejše informacije glej [25])  |
| #<data>            | takošnji celoštevilčni podatek  |
| oznaka             | oznaka (simbolični naslov) v zbirniku (angl. label)   |
| list               | lista, seznam registrov, npr. D3-D0   |
| [...]              | biti operanda, npr. [7] označuje 7. bit, [32:24] pa bite od 32 do 24  |
| (...)              | vsebina lokacije, npr. (Rn) označuje vsebino registra Rn  |
| CCR                | register za pogojno kodo (angl. condition code register) - spodnji byte statusnega registra (sestavljen iz bitov X, N, Z, V, C; glej pogl. 4.1.2) |
| PC                 | programskega števca   |
| SP                 | aktivni kazalec sklada  |
| SR                 | statusni register   |
| SSP                | kazalec sklada v nadzornem režimu   |
| USP                | uporabniški kazalec sklada  |
| FC                 | funkcijska koda   |
| DFC                | register za funkcijsko kodo končnega naslova  |
| SFC                | register za funkcijsko kodo začetnega naslova   |

<sup>5</sup> Primer: pri ukazu ADD.L D0,D1 je D0 Source, D1 pa Dest. To pomeni, da bo seštevek vsebin obeh registrov shranjen v registru D1, vsebina D0 pa ostane nespremenjena.

- Booleova logična funkcija IN
- + Booleova logična funkcija ALI
- $\oplus$  Booleova logična funkcija EKSCLUZIVNI ALI
- $\underline{x}$  Booleov logični komplement (invertiranje operanda)
- LSW, MSW manj pomembna (desna) in bolj pomembna (leva) beseda

V tabelah ukazov vsebuje prvi stolpec t.i. *mnemonik* ukaza<sup>6</sup>, drugi splošno sintakso oz. dovoljene tipe operanda (ali operandov). Tretji stolpec deklarira možne tipe oz. dolžine operandov (B, W, L: 8, 16 ali 32 bitov), četrти pa opisuje funkcijo ukaza.

## OPOZORILO!

Načeloma velja, da nadomešča simbol <ea> katerikoli tip naslova: Dx, Ax, #xxx itd. Za omejitve pri posameznih ukazih glej literaturo [25].

Primeri ukazov so pisani v debug/monitorskem programu CPU32BUG, kjer so z “%” označena binarna števila, z “&” desetiška, operandi s simbolom “\$” oz. brez kakršnekoli oznake pa so pisani v šestnajstistiški kodri. Tukaj velja še povedati, da se pri nekaterih drugih zbirnikih sintaksa nekoliko razlikuje od tukaj prikazane.

### 4.2.1 Ukazi za prenos podatkov

| Ukaz  | Sintaksa                                 | Dolžina operanda                  | Operacija  |
|-------|--|-----------------------------------|--|
| EXG   | Rn, Rn                                   | 32                                | Rn $\Rightarrow$ Rn  |
| LEA   | <ea>, An                                 | 32                                | <ea> $\Rightarrow$ An  |
| LINK  | An, #<d>                                 | 16, 32                            | SP-4 $\Rightarrow$ SP, An $\Rightarrow$ (SP); SP $\Rightarrow$ An, SP+d $\Rightarrow$ SP   |
| MOVE  | <ea>, <ea>                               | 8, 16, 32                         | Source $\Rightarrow$ Dest  |
| MOVEA | <ea>, An                                 | 16, 32 $\Rightarrow$ 32           | Source $\Rightarrow$ Dest  |
| MOVEM | list, <ea><br><ea>, list                 | 16, 32<br>16, 32 $\Rightarrow$ 32 | našteti registri $\Rightarrow$ Dest<br>Source $\Rightarrow$ našteti registri   |
| MOVEP | Dn, (d16,<br>An)<br><br>(d16, An),<br>Dn | 16, 32                            | Dn[31:24] $\Rightarrow$ (An+d); Dn[23:16] $\Rightarrow$ (An+d+2);<br>Dn[15:8] $\Rightarrow$ (An+d+4);<br>Dn[7:0] $\Rightarrow$ (An+d+6)<br><br>(An+d) $\Rightarrow$ Dn[31:24]; (An+d+2) $\Rightarrow$ Dn[23:16];<br>(An+d+4) $\Rightarrow$<br>Dn[15:8]; (An+d+6) $\Rightarrow$ Dn[7:0] |
| MOVEQ | #<data>, Dn                              | 8 $\Rightarrow$ 32                | takojšnji podatek $\Rightarrow$ Dest   |
| PEA   | <ea>                                     | 32                                | SP-4 $\Rightarrow$ SP; <ea> $\Rightarrow$ SP   |
| UNLK  | An                                       | 32                                | An $\Rightarrow$ SP; (SP) $\Rightarrow$ An, SP+4 $\Rightarrow$ SP  |

Osnovna funkcija te skupine ukazov je prenos in shranjevanje podatkov in naslovov (npr. iz registra v register ali pomnilnik).

<sup>6</sup> Izraz “mnemonik” ali “mnemotehnični” pomeni, da nas že naziv ukaza spominja na njegovo funkcijo, npr. ukaz CLR (iz angleške besede “clear” - izbrisati) postavi vse bite operanda v stanje “0”.

**Primeri ukazov:**

- **MOVEA.L #\\$12345678, A3**

32-bitno konstanto 12345678<sub>HEX</sub> naloži v A3.

- **MOVE.B D0, D2**

pred izvršitvijo: (D0)=12345678<sub>HEX</sub> in (D2)=87654321<sub>HEX</sub>,  
po izvršitvi: (D0) = 12345678<sub>HEX</sub> in (D2)=87654378<sub>HEX</sub>.

**PAZI:** le spodnji byte ciljne lokacije se zamenja s spodnjim bytom začetnega naslova!

### 4.2.2 Ukazi za celoštevilčno (integer) aritmetiko

| Ukaz        | Sintaksa                               | Dolžina operanda                       | Operacija  |
|-------------|--|--|--|
| ADD         | Dn, <ea> &lt;ea&gt;, Dn</ea>           | 8, 16, 32<br>8, 16, 32                 | Source+Dest⇒Dest   |
| ADDA        | <ea>, An                               | 16, 32                                 | Source+Dest⇒Dest   |
| ADDI        | #<data>,<ea>                           | 8, 16, 32                              | takošnji podatek+Dest⇒Dest                               |
| ADDQ        | #<data>,<ea>                           | 8, 16, 32                              | takošnji podatek+Dest⇒Dest                               |
| ADDX        | Dn, Dn<br>-(An), -(An)                 | 8, 16, 32<br>8, 16, 32                 | Source+Dest+X⇒Dest                                       |
| CLR         | <ea>                                   | 8, 16, 32                              | 0⇒Dest   |
| CMP         | <ea>, Dn                               | 8, 16, 32                              | (Dest-Source), postavi CCR                               |
| CMPA        | <ea>, An                               | 16, 32                                 | (Dest-Source), postavi CCR                               |
| CMPI        | #<data>,<ea>                           | 8, 16, 32                              | (Dest-data), postavi CCR                                 |
| CMPM        | (An)+, (An)+                           | 8, 16, 32                              | (Dest-Source), postavi CCR                               |
| CMP2        | <ea>, Rn                               | 8, 16, 32                              | Sp. meja ≤ Rn ≤ Zg. meja, postavi CCR                    |
| DIVS/DIVU   | <ea>, Dn                               | 32/16⇒16:16                            | Dest/Source⇒Dest (z ali brez predznaka)                  |
| DIVSL/DIVUL | <ea>, Dr:Dq<br><ea>, Dq<br><ea>, Dr:Dq | 64/32⇒32:32<br>32/32⇒16<br>32/32⇒32:32 | Dest/Source⇒Dest (z ali brez predznaka)                  |
| EXT         | Dn<br>Dn                               | 8⇒16<br>16⇒32                          | razširitev Dest s predznakom ⇒Dest                       |
| EXTB        | Dn                                     | 8⇒32                                   | razširitev Dest s predznakom ⇒Dest                       |
| MULS/MULU   | <ea>, Dn<br><ea>, D1<br><ea>, Dh:D1    | 16*16⇒32<br>32*32⇒32<br>32*32⇒64       | Dest*Source⇒Dest (z ali brez predznaka)                  |
| NEG         | <ea>                                   | 8, 16, 32                              | 0-Dest⇒Dest  |
| NEGX        | <ea>                                   | 8, 16, 32                              | 0-Dest-X⇒Dest  |
| SUB         | <ea>, Dn<br>Dn, <ea>                   | 8, 16, 32                              | Dest-Source⇒Dest   |
| SUBA        | <ea>, An                               | 16, 32                                 | Dest-Source⇒Dest   |
| SUBI        | #<data>,<ea>                           | 8, 16, 32                              | Dest-data⇒Dest   |
| SUBQ        | #<data>,<ea>                           | 8, 16, 32                              | Dest-data⇒Dest   |
| SUBX        | Dn, Dn<br>-(An), -(An)                 | 8, 16, 32<br>8, 16, 32                 | Dest-Source-X⇒Dest                                       |
| TBLS/TBLU   | <ea>, Dn<br>Dym:Dyn, Dn                | 8, 16, 32                              | Dyn-Dym⇒Temp<br>(Temp*Dn[7:0])⇒Temp<br>(Dym*256)+Temp⇒Dn |
| TBLSN/TBLUN | <ea>, Dn<br>Dym:Dyn, Dn                | 8, 16, 32                              | Dyn-Dym⇒Temp<br>(Temp*Dn[7:0])/256⇒Temp<br>Dym+Temp⇒Dn   |

Poleg enostavnejših aritmetičnih operacij z različnimi tipi operandov (seštevanje, odštevanje, brisanje, negiranje...), sodita v to skupino še ukaza za množenje in deljenje, ki običajno nista del nabora ukazov enostavnejših procesorjev.

Ukazi za primerjavo dveh operandov so v osnovu operacije odštevanja, pri čemer se oba operanda ne spremenita. Edina posledica ukaza je postavitev ustreznih bitov v CCR registru.

#### Primeri ukazov:

- **NEG.L D0**

pred ukazom: (D0) = 00001234<sub>HEX</sub>,  
po ukazu: (D0) = fffffedcc<sub>HEX</sub>

- **ADD.B D3,D0**

pred ukazom: (D3) = 12121212<sub>HEX</sub>, (D0) = 12345678<sub>HEX</sub>,  
po ukazu: (D3) = 12121212<sub>HEX</sub>, (D0) = 1234568a<sub>HEX</sub> (sodeluje le LSByte).

- **SUBI.L #fffffff,D0**

pred ukazom: (D0) = fffffff<sub>HEX</sub> = -1<sub>DEC</sub>,  
po ukazu: (D0) = 0, zaradi: -1<sub>DEC</sub> - (-1<sub>DEC</sub>) = 0.

#### 4.2.3 Logične operacije

| Ukaz | Sintaksa             | Dolžina operanda       | Operacija              |
|------|----------------------|------------------------|------------------------|
| AND  | <ea>, Dn<br>Dn, <ea> | 8, 16, 32<br>8, 16, 32 | Source•Dest⇒Dest       |
| ANDI | #<data>, <ea>        | 8, 16, 32              | data•Dest⇒Dest         |
| EOR  | Dn, <ea>             | 8, 16, 32              | Source⊕Dest⇒Dest       |
| EORI | #<data>, <ea>        | 8, 16, 32              | data⊕Dest⇒Dest         |
| NOT  | <ea>                 | 8, 16, 32              | Dest⇒Dest              |
| OR   | <ea>, Dn<br>Dn, <ea> | 8, 16, 32<br>8, 16, 32 | Source+Dest⇒Dest       |
| ORI  | #<data>, <ea>        | 8, 16, 32              | data+Dest⇒Dest         |
| TST  | <ea>                 | 8, 16, 32              | Source-0; spremeni CCR |

V tej skupini so ukazi za izvajanje logičnih operacij IN, ALI, EKSCLUZIVNI ALI in NE (angl. AND, OR, EOR in NOT) ter njihove inačice z neposrednim operandom. V operacijah se povezujejo biti na istih pozicijah (enake uteži) znotraj operandova.

#### Primeri ukazov:

- **AND.L (A0), D0**

pred ukazom:

- $(A0) = 00008000_{HEX}$ ,
- dvojna beseda na naslovu (8000) je  $87654321_{HEX}$  ter
- $(D0) = \text{ffffedcc}_{HEX}$ ;

po ukazu:

- $(A0) = 00008000_{HEX}$ ,
- $(8000) = 87654321_{HEX}$  ter
- $(D0) = 87654100_{HEX}$  (zaradi lažje ponazoritve pretvori v binarna števila).

- **EORI.W #\\$1212, D0**

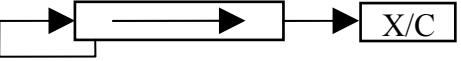
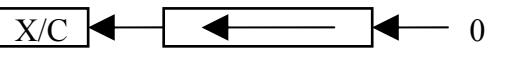
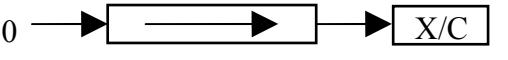
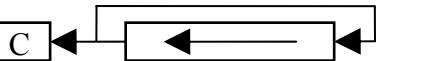
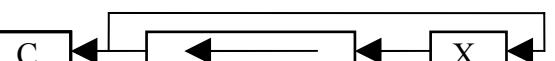
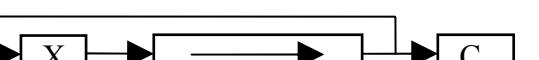
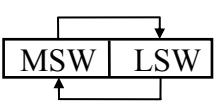
pred ukazom:  $(D0) = 33333333_{HEX}$ ,

po ukazu:  $(D0) = 33332121_{HEX}$  (sodeluje le LSW):

- **TST.W D0**

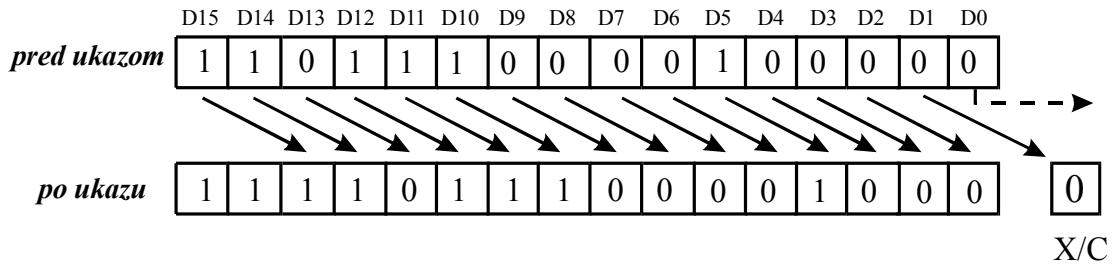
$(D0) = 1234abcd_{HEX}$  pred in po izvršitvi ukaza; ukaz postavi bit N (rezultat negativen) v CCR (del statusnega registra), saj obravnava le LSW ( $abcd_{HEX}$ ), ki je negativen po logiki dvojiškega komplementa:

#### 4.2.4 Pomične in rotacijske operacije

| Ukaz | Sintaksa                                | Dolžina operanda             | Operacija  |
|------|---|------------------------------|--|
| ASL  | $D_n, D_n$<br>$\#<data>, D_n$<br>$<ea>$ | 8, 16, 32<br>8, 16, 32<br>16 |    |
| ASR  | $D_n, D_n$<br>$\#<data>, D_n$<br>$<ea>$ | 8, 16, 32<br>8, 16, 32<br>16 |    |
| LSL  | $D_n, D_n$<br>$\#<data>, D_n$<br>$<ea>$ | 8, 16, 32<br>8, 16, 32<br>16 |    |
| LSR  | $D_n, D_n$<br>$\#<data>, D_n$<br>$<ea>$ | 8, 16, 32<br>8, 16, 32<br>16 |    |
| ROL  | $D_n, D_n$<br>$\#<data>, D_n$<br>$<ea>$ | 8, 16, 32<br>8, 16, 32<br>16 |    |
| ROR  | $D_n, D_n$<br>$\#<data>, D_n$<br>$<ea>$ | 8, 16, 32<br>8, 16, 32<br>16 |  |
| ROXL | $D_n, D_n$<br>$\#<data>, D_n$<br>$<ea>$ | 8, 16, 32<br>8, 16, 32<br>16 |  |
| ROXR | $D_n, D_n$<br>$\#<data>, D_n$<br>$<ea>$ | 8, 16, 32<br>8, 16, 32<br>16 |  |
| SWAP | $D_n$                                   | 16                           |  |

V tej skupini so ukazi za pomik (angl. shift) bitov znotraj operanda. Ukazi se razlikujejo glede na smer in število mest pomika, stanje "izpraznjenega" bita (MSB ali LSB) ter položaj ali indikacijo "izpadlega" bita.

Kot primer si lahko ogledamo ukaz **A\$R** • W #2, D0 (slika 4. 3). Pomik se nanaša le na spodnjih 16 bitov registra D0. Predpostavimo, da obdelujemo le 16-bitne informacije, torej je pred izvršitvijo ukaza vsebina spodnje besede v D0 po predznaku negativna: 1101 1100 0010 0000<sub>BIN</sub> = dc20<sub>HEX</sub> = -9184<sub>DEC</sub> (MSW = 0). Ukaz premakne vse bite v registru za dve poziciji v desno, pri čemer se prazni poziciji na levi naložita z "1" (MSB), skrajnji desni biti pa zapustijo register (razen zadnjega, ki se naloži v X oz. C bit statusnega registra). Nova vsebina registra je 1111 0111 0000 1000<sub>BIN</sub> = f708<sub>HEX</sub> = -2296<sub>DEC</sub> (ob upoštevanju le spodnje besede), torej štirikrat manjša od izhodiščne. Rezultat je pričakovani, saj je pomik bitov v levo (npr. z ukazom **ASL**) ali v desno za  $n$  pozicij enak množenju oz. deljenju z  $2^n$ .

**Slika 4. 3: Pomik bitov pri ukazu ASR.W #2, D0**

Podoben rezultat ukaza ASR dosežemo tudi z ukazom LSR, obstaja pa ena bistvena razlika: pri LSR ukazu se pri vsakem pomiku za en bit MSB napolni vedno z "0", pri ASR pa se preslika prejšnji MSB. Slednji torej ohranja predznak originalnega števila in opravlja deljenja z 2, 4, 8 itd., LSR pa le premika bite, brez upoštevanja predznaka (od tod tudi naziv "logični premik v desno", Logical Shift Right - LSR).

#### Primeri ukazov:

Naslednja skupina ukazov obdeluje D0, katerega vsebina pred izvršitvijo je enaka  $12345678_{\text{HEX}} = 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000_{\text{BIN}}$ . Komentar kaže spremenjena stanja D0:

- ROL.B #1, D0

$123456F0_{\text{HEX}} = 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 1111\ 0000_{\text{BIN}}$ ;

biti desnega byta se premaknejo za eno pozicijo v levo, MSB pa postane LSB; ostali biti v D0 so nespremenjeni.

- LSL.L #2, D0

$48D15BC0_{\text{HEX}} = 0100\ 1000\ 1101\ 0001\ 0101\ 1011\ 1100\ 0000_{\text{BIN}}$ ;

celotna vsebina D0 se premakne v levo za dva bita; pri vsakem premiku se c LSB naloži 0.

- SWAP D0

$5BC048D1_{\text{HEX}} = 0101\ 1011\ 1100\ 0000\ 0010\ 1000\ 1101\ 0001_{\text{BIN}}$ ; LSW in MSW zamenjata mesti.

#### 4.2.5 Operacije manipuliranja z biti

| Ukaz | Sintaksa   | Dolžina operanda | Operacija   |
|------|--|------------------|---|
| BCHG | $\text{Dn}, \langle ea \rangle, \# \langle data \rangle, \langle ea \rangle$ | 8, 32            | $(\langle \text{št. bita} \rangle \text{ Dest}) \Rightarrow Z \Rightarrow \text{bit v Dest}$        |
| BCLR | $\text{Dn}, \langle ea \rangle, \# \langle data \rangle, \langle ea \rangle$ | 8, 32            | $(\langle \text{št. bita} \rangle \text{ Dest}) \Rightarrow Z$<br>$0 \Rightarrow \text{bit v Dest}$ |
| BSET | $\text{Dn}, \langle ea \rangle, \# \langle data \rangle, \langle ea \rangle$ | 8, 32            | $(\langle \text{št. bita} \rangle \text{ Dest}) \Rightarrow Z$<br>$1 \Rightarrow \text{bit v Dest}$ |
| BTST | $\text{Dn}, \langle ea \rangle, \# \langle data \rangle, \langle ea \rangle$ | 8, 32            | $(\langle \text{št. bita} \rangle \text{ Dest}) \Rightarrow Z$                                      |

Naloga te skupine ukazov je spreminjanje in testiranje posameznih bitov znotraj byta ali dolge besede.

#### Primeri ukazov:

Vsebina D0 je pred izvajanjem zaporedja ukazov:

$$1111\ 1111_{\text{HEX}} = 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001\ 0001_{\text{BIN}}.$$

- **BCHG.L #3,D0**

stanje bita 3 v D0 (četrtega z desne strani) je 0, zato setiraj Z bit CCR, nato pa isti bit v D0 negiraj; rezultat je  $1111\ 1119_{\text{HEX}}$ .

- **BCLR.L #3,D0**

isti bit najprej primerjaj z “0”, ker pa je “1”, resetiraj Z v CCR; nato briši bit v D0; rezultat je  $1111\ 1111_{\text{HEX}}$ .

#### 4.2.6 Operacije z BCD števili

| Ukaz               | Sintaksa  | Dolžina operanda | Operacija  |
|--------------------|---|------------------|--|
| ABC $\overline{D}$ | D <sub>n</sub> , D <sub>n</sub> - (A <sub>n</sub> ) <sub>1</sub> - (A <sub>n</sub> ) <sub>2</sub> | 8                | Source <sub>10</sub> + Dest <sub>10</sub> + X $\Rightarrow$ Dest |
| NBC $\overline{D}$ | <ea>  | 8                | 0 - Dest <sub>10</sub> - X $\Rightarrow$ Dest                    |
| SBC $\overline{D}$ | D <sub>n</sub> , D <sub>n</sub> - (A <sub>n</sub> ) <sub>1</sub> - (A <sub>n</sub> ) <sub>2</sub> | 8                | Dest <sub>10</sub> - Source <sub>10</sub> - X $\Rightarrow$ Dest |

V poglavju 1.1.3 smo videli, da je BCD format nekakšen vmesni zapis med decimalnim in binarnim. Vsak naslednji levi nibble množimo z naslednjo višjo potenco baze 10, saj ustrezajo enicam, deseticam, stoticam itd. Zato veljajo za takšen zapis tudi posebna aritmetična pravila. Pri procesorjih, ki teh ukazov nimajo, je treba napisati posebne podprograme za BCD operacije, kot so seštevanje, odštevanje in negacija. Ukazi obravnavajo le LSByte.

**Primer ukaza:**

- ABC $\overline{D}$  D $\overline{0}$ , D $\overline{1}$

Začetni pogoji: (D0) = 12345678<sub>HEX</sub>, (D1) = 33333333<sub>HEX</sub>.  
Po ukazu: (D1) = 33333311<sub>HEX</sub>, razširitveni bit X = “1”.

**Pojasnilo:**

Skrajna desna nibbla seštevamo enako kot decimalna števila. X bit v CCR se postavi v stanje 1, če rezultat presega obseg enega byta:

$$\begin{array}{r}
 78_{\text{DEC}} \\
 + 33_{\text{DEC}} \\
 \hline
 111_{\text{DEC}}
 \end{array}$$

setiraj bit X

Rezultat “navadnega” binarnega seštevanja bi bil ab<sub>HEX</sub>.

#### 4.2.7 Ukazi za nadzor nad potekom izvajanja programa

| Ukaz                           | Sintaksa     | Dolžina operanda | Operacija  |
|--------------------------------|--------------|------------------|--|
| <b>Pogojno izvajanje</b>       |              |                  |  |
| Bcc                            | <oznaka>     | 8, 16, 32        | če pogoj <i>TRUE</i> , PC + d $\Rightarrow$ PC   |
| DBcc                           | Dn, <oznaka> | 16               | če pogoj <i>FALSE</i> , Dn - 1 $\Rightarrow$ PC;<br>če Dn $\neq$ (-1), PC + d $\Rightarrow$ PC     |
| Scc                            | <ea>         | 8                | če pogoj <i>TRUE</i> , setiraj bite v Dest; drugače pa jih resetiraj                               |
| <b>Brezpogojno izvajanje</b>   |              |                  |  |
| BRA                            | <oznaka>     | 8, 16, 32        | PC + d $\Rightarrow$ PC  |
| BSR                            | <oznaka>     | 8, 16, 32        | SP - 4 $\Rightarrow$ SP; PC $\Rightarrow$ (SP); PC + d $\Rightarrow$ PC                            |
| JMP                            | <ea>         |                  | Dest $\Rightarrow$ PC  |
| JSR                            | <ea>         |                  | SP - 4 $\Rightarrow$ SP; PC $\Rightarrow$ (SP); Dest $\Rightarrow$ PC                              |
| NOP                            |              |                  | PC + 2 $\Rightarrow$ PC  |
| <b>Vrnitve iz podprogramov</b> |              |                  |  |
| RTD                            | #<d>         | 16               | (SP) $\Rightarrow$ PC; SP + 4 + d $\Rightarrow$ SP   |
| RTR                            |              |                  | (SP) $\Rightarrow$ CCR; SP + 2 $\Rightarrow$ SP; (SP) $\Rightarrow$ PC;<br>SP + 4 $\Rightarrow$ SP |
| RTS                            |              |                  | (SP) $\Rightarrow$ PC; SP + 4 $\Rightarrow$ SP   |

Pogoj za skok pri pogojnih skokih izberemo tako, da oznako "cc" iz prejšnje tabele (ukazi Bcc, DBcc in Scc) zamenjamo z naslednjimi simboli:

|    |   |    |   |
|----|---|----|---|
| CC | - C resetiran                           | CS | - C setiran                                       |
| EQ | - enak (angl. equal)                    | T  | - TRUE (trditev resnična)                         |
| GE | - večji (angl. greater) ali enak        | F  | - FALSE (trditev neresnična - le pri DBcc in Scc) |
| HI | - višji (angl. higher)                  | GT | - večji od  |
| LE | - manjši ali enak (angl. less or equal) | LT | - manjši od                                       |
| LS | - nižji ali enak (angl. low or same)    | NE | - neenak  |
| MI | - negativen                             | VC | - bit V v SP (presežek) resetiran                 |
| PL | - pozitiven                             | VS | - V setiran                                       |

*Tabela 4. 1: Simboli za pogoje pri pogojnih skokih (vpišemo jih namesto cc)*

Tukaj moramo opozoriti na razliko med pogojema "večji" in "manjši" ter "višji" in "nižji". Pogoja "večji" in "manjši" (ukazi GE, GT, LT in LE) pomenita, da se vsebini prizrejenih operandov interpretirata kot predznačena števila (S). Pri pogojih "višji" in "nižji" (ukaza HI, LS) pa se vsebini operandov obravnavata kot nepredznačena števila (U, glej tudi pogl. 1.2).

Včasih je treba linearno izvajanje programov nadaljevati z ukazom na neki drugi lokaciji. Tak skok je lahko *brezpogojen* ali pa odvisen od rezultata prejšnje operacije oz. od stanja bitov v statusnem registru - *pogojni skok* (slika 4. 4).

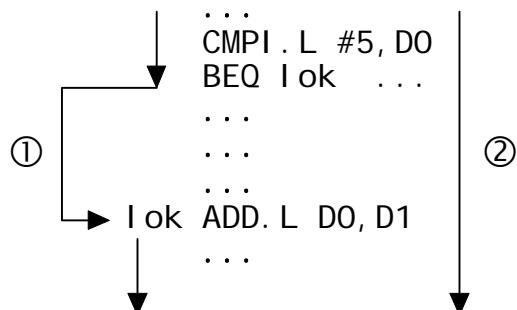
Poleg tega delimo skoke v dve veliki skupini:

- **enostavni skoki na lokacije znotraj podprograma,**
- **skoki na podprograme.**

V prvo skupino sodijo skoki, pri katerih se spremeni samo vsebina PC, vsi ostali registri pa ostanejo nespremenjeni. Druga skupina skokov je predvidena za nadaljevanje programa na ločeni programski sekvenci (t.i. *podprogramu*, angl. subroutine), po koncu katere se bomo vrnili na začetno skočno lokacijo. Pri tem se poleg PC spremeni tudi vsebina SP.

#### 4.2.7.1 Enostavni skoki na lokacije

Slika 4. 4 kaže primer enostavnega pogojnega skoka na lokacijo, ki smo jo označili s simbolom (angl. label). V konkretnem primeru se bo izvajanje programa nadaljevalo z ukazom, ki se nahaja na lokaciji **l ok** (sekvenca ①), če bo izpolnjen pogoj, da je setiran bit Z v SR (vsebina D0 je enaka 5). V nasprotnem primeru se bo program izvajal linearno (sekvenca ②). Pomembno je vedeti, da se bo tudi v tem primeru ukaz na oznaki **l ok** izvajal po izvršitvi predhodnih ukazov, če med njimi ni kakega drugega skoka. Pri brezpogojnih skokih (**BRA** in **JMP**) se sekvenca ① izvaja vedno, ne glede na rezultat prejšnje operacije.



**Slika 4. 4: Primer enostavnega brezpogojnega skoka na lokacijo**

Načeloma obstajata dva tipa ukazov za skoke oz. razvejitve: prvi imajo začetnico "J" (iz angl. jump - skok; npr. **JMP**), drugi pa "B" (iz angl. branch - razvejitev; npr. **BCC**). Pri prvem se naslov skočne lokacije nahaja v registru, ki je določen v 6-bitnem polju  $\langle ea \rangle^7$  16-bitne kode ukaza. V drugem primeru skočno lokacijo dobimo tako, da trenutni vrednosti PC pristejemo odmik (t.i. "displacement" - d). 8-bitni odmik (skočni naslov  $PC \pm 127$ ), je sestavni del kode ukaza, za določanje 16- ali 32-bitnega odmika pa rabimo eno oz. dve dodatni besedi..

#### Primer ukazov:

<sup>7</sup> Navadno jo označujemo s simboličnim naslovom.

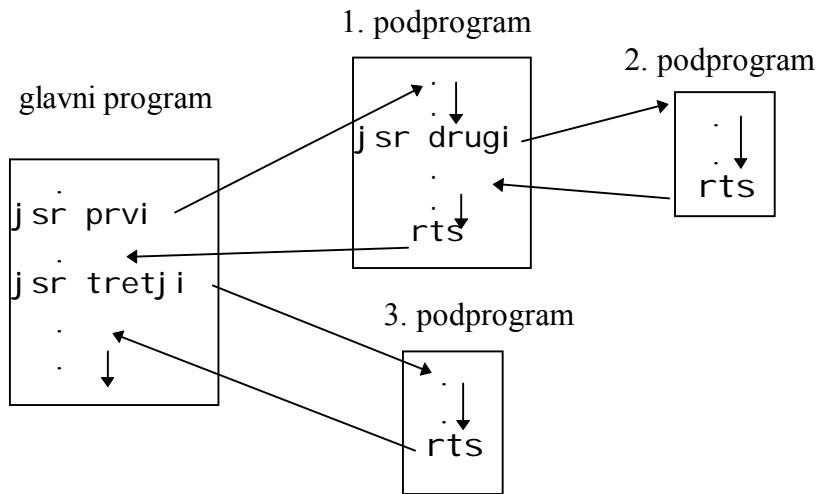
Vsebina D0 je c000 0000<sub>HEX</sub>. Po primerjavi D0 z 0 (npr. ukaz **CMP.L #0,D0**) bo pogoj za skok **BHI skok** izpolnjen, za **BGT skok** pa ne. Velja namreč:

nepredznačeno (U): c000 0000<sub>HEX</sub> = 3 221 225 472<sub>DEC</sub> (večji od nič),  
predznačeno (S): c000 0000<sub>HEX</sub> = -1 073 741 824<sub>DEC</sub> (manjši od nič).

#### 4.2.7.2 Skoki na podprograme

Zapletene in dolge programe, kjer želimo ohraniti preglednost nad potekom izvajanja, ali programe, pri katerih med izvajanjem večkrat ponovimo enako programsko sekvenco<sup>8</sup>, je smiselno sestavljati modularno, v podprogramih ali funkcijah. Takrat govorimo o *strukturiranem programiraju* (slika 4. 5).

Za skoke na podprogram je značilna načelna vrnitev v glavni program ali podprogram, iz katerega smo izvršili prvotni skok. Izvajanje programa se nadaljuje z ukazom, ki sledi skočnemu ukazu. Iz enega podprograma lahko skočimo na naslednjega, iz njega na naslednjega itd. Ta postopek imenujemo *gnezdenje* (angl. nesting), kar kaže slika 4. 5. Število nivojev gnezdenja je načeloma omejeno z velikostjo sklada.



Slika 4. 5: Blokovna shema strukturnega programiranja

Iz opisanega sledita dva osnovna koraka pri uporabi podprogramov (ne glede na to, ali imamo opravka z razvezitvijo ali s skokom):

- **skok na podprogram (BSR ali JSR)**

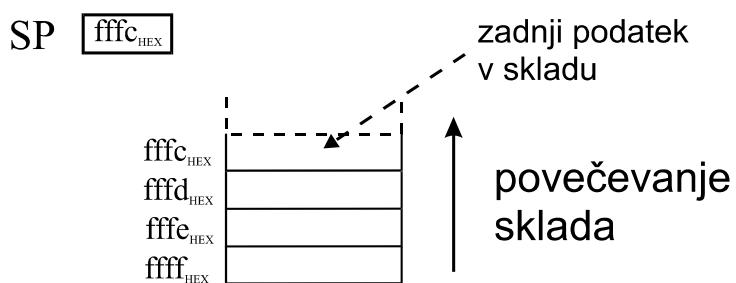
<sup>8</sup> Pri tem so lahko vhodni podatki v to sekvenco različni, ukazi pa vedno enaki. Npr. sekvenco za PI regulator v regulacijskem algoritmu enosmerjnega motorja lahko uporabimo za regulacijo hitrosti, fluksa ali toka.

1. kazalec sklada (SP) se zmanjša za štiri (sprosti prostor štirih bytov - 32 bitov - v skladu), ker se
2. vrednost tekočega PC (32-bitni naslov ukaza za skočnim ukazom v starem podprogramu) naloži v sklad,
3. v PC se naloži začetna lokacija podprograma (nadaljevanje izvajanja ukazov na podprogramske modulu).

- **vrnitev iz podprograma** (ukaz RT<sub>S</sub> - angl. Return from Subroutine):
  1. vsebina lokacije sklada na katero kaže SP se naloži v PC (stara vsebina PC, ki kaže na naslednji ukaz za skokom v glavnem programu),
  2. kazalec sklada se vrne na prejšnjo vrednost (se poveča za štiri).

Kot vidimo, je ena od funkcij sklada začasno shranjevanje sistemskih podatkov (v tem primeru naslov, s katerim nadaljujemo izvajanje programa po vrnitvi iz podprograma). Pri izvajjanju ukazov, kot so npr. JSR, BS<sub>R</sub>, RT<sub>S</sub> itd., se vrednost kazalca sklada, ki kaže na zadnji naloženi podatek, samodejno zmanjšuje (skok na podprogram) ali povečuje (vrnitev iz podprograma)<sup>9</sup>. Pri tem moramo paziti na naslednje:

- Če med skokom na podprogram in vrnitvijo iz njega (komplementarna ukaza) želimo nekaj naložiti v sklad oz. spremeniti vrednost v SP, moramo pred ukazom RT<sub>S</sub> vrniti SP v izhodiščno stanje. V nasprotnem se na vrhu sklada ne nahaja povratni naslov, zato ne preberemo pravilnega podatka in je vrnitev onemogočena. Na pravilno manipuliranje s skladom moramo biti še posebej pozorni pri gnezdenju.
- Za sklad moramo rezervirati področje v delovnem RAM pomnilniku. Njegova velikost je odvisna od zahtev programa (npr. od nivojev gnezdenja). Prav zaradi tega je definirana dinamično. Sklad se običajno postavi na **najvišji prosti naslov RAM pomnilnika**. Polnjenje sklada oz. njegovo povečevanje poteka proti nižjim lokacijam, torej se pri polnjenju sklada SP zmanjšuje (slika 4. 6). Edina skrb programerja je, da se sklad in ostali koristni podatki ali program v RAM tudi pri še takoj velikem številu lokacij sklada nikoli ne prekrivajo!



*Slika 4. 6: Smer polnjenja sklada*

#### 4.2.8 Ukazi za sistemski nadzor

<sup>9</sup> Sklad deluje na LIFO principu (angl. Last In First Out): zadnji vhodni podatek bo tudi prvi izhodni.

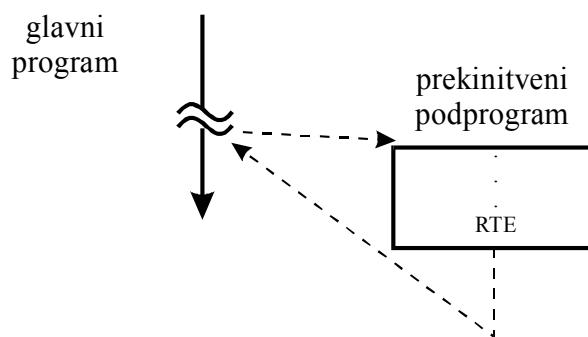
| Ukaz   | Sintaksa  | Dolžina operand a | Operacija   |
|--|---|-------------------|---|
| <b>Privilegirani</b>                                   |   |                   |   |
| ANDI   | #<data>, SR                                       | 16                | Data • SR $\Rightarrow$ SR  |
| EORI   | #<data>, SR                                       | 16                | Data $\oplus$ SR $\Rightarrow$ SR   |
| MOVE   | <ea>, SR<br>SR, <ea>                              | 16<br>16          | Source $\Rightarrow$ SR<br>SR $\Rightarrow$ Dest  |
| MOVEA  | USP, An<br>An, USP                                | 32<br>32          | USP $\Rightarrow$ An<br>An $\Rightarrow$ USP  |
| MOVEC  | Rc, Rn<br>Rn, Rc                                  | 32<br>32          | Rc $\Rightarrow$ Rn<br>Rn $\Rightarrow$ Rc  |
| MOVES  | Rn, <ea><br><ea>, Rn                              | 8, 16, 32         | Rn $\Rightarrow$ Dest ob uporabi DFC<br>Source ob uporabi SFC $\Rightarrow$ Rn  |
| ORI  | #<data>, SR                                       | 16                | Data + SR $\Rightarrow$ SR  |
| RESET  |   |                   | aktiviraj linijo RESET  |
| RTE  |   |                   | (SP) $\Rightarrow$ SR, SP + 2 $\Rightarrow$ SP; (SP) $\Rightarrow$ PC,<br>SP + 4 $\Rightarrow$ SP; ponovno vzpostavi sklad ustrezno formatu   |
| STOP   | #<data>   | 16                | Data $\Rightarrow$ SR; STOP   |
| LPSTOP   | #<data>   |                   | Data $\Rightarrow$ SR; prekinitenvena maska $\Rightarrow$ EBI; STOP   |
| <b>Generiranje TRAP funkcije</b>                       |   |                   |   |
| BKPT   | #<data>   |                   | če prekinitveni (breakpoint) ciklus potren, izvrši vrnjeno ukazno besedo, drugače izvrši trap funkcijo zaradi ilegalnega ukaza  |
| BGND   |   |                   | če bacground modus omogočen, izvrši background modus, drugače format/vektor odmik $\Rightarrow$ - (SSP);<br>PC $\Rightarrow$ - (SSP); SR $\Rightarrow$ -(SSP); (vektor) $\Rightarrow$ PC                              |
| CHK  | <ea>, Dn  | 16, 32            | če Dn < 0 OR Dn < (ea), CHK prekinitev  |
| CHK2   | <ea>, Rn  | 8, 16, 32         | če Rn < nižja meja OR Rn > višja meja, CHK prekinitev   |
| ILLEGAL  |   |                   | SSP - 2 $\Rightarrow$ SSP; vektor odmik $\Rightarrow$ (SSP);<br>SSP - 4 $\Rightarrow$ SSP; PC $\Rightarrow$ (SSP);<br>SSP - 2 $\Rightarrow$ SSP; SR $\Rightarrow$ (SSP);<br>vektor nelegalnega ukaza $\Rightarrow$ PC |
| TRAP   | #<data>   |                   | SSP - 2 $\Rightarrow$ SSP; format/vektor odmik $\Rightarrow$ (SSP)<br>SSP - 4 $\Rightarrow$ SSP; PC $\Rightarrow$ (SSP), SR $\Rightarrow$ (SSP);<br>naslov vektorja $\Rightarrow$ PC                                  |
| TRAPcc   | ni <input checked="" type="checkbox"/><br>#<data> | 16, 32            | če cc TRUE, TRAP prekinitev   |
| TRAPV  |   |                   | če V setiran, TRAP prekinitev zaradi presežka   |
| <b>Register CCR (spodnji byte statusnega registra)</b> |   |                   |   |
| ANDI   | #<data>, CCR                                      | 8                 | Data • CCR $\Rightarrow$ CCR  |
| EORI   | #<data>, CCR                                      | 8                 | Data $\oplus$ CCR $\Rightarrow$ CCR   |
| MOVE   | <ea>, CCR<br>CCR, <ea>                            | 16<br>16          | Source $\Rightarrow$ CCR<br>CCR $\Rightarrow$ Dest  |
| ORI  | #<data>, CCR                                      | 8                 | Data + CCR $\Rightarrow$ CCR  |

Zgornja skupina ukazov vpliva na sistemski nadzor. Npr. ukaz STOP ustavi nadaljnjo obdelavo programa, enako kot LPSTOP, ki dodatno postavi mikrokrmilnik v čakalno stanje

minimalne porabe<sup>10</sup> do nastopa prekinitve, ukaza TRACE ali resetiranja mikroračunalnika. Tukaj bomo največjo pozornost posvetili ukazu RTE (angl. Return from Exception), ki je potreben za vrnitev iz prekinitvenega podprograma.

#### 4.2.8.1 Prekinitve pri MC68332

*Prekinitve* (angl. interrupts) so najhitrejši način programskega servisiranja nekega izjemnega dogodka. Po definiciji je to asinhronski dogodek, ki ustavi normalno izvajanje programske sekvence in začasno usmeri potek izvajanja programa na posebno programsko sekvenco (*prekinitveni podprogram*, angl. interrupt handler ali subroutine). Velikokrat se namreč zgodi, da je treba čim hitreje ukrepati na nek dogodek (npr. indikacija nadtokovne zaščite signalizira preobremenjenost motorja). Ukaz, s katerim bi ob normalnem poteku izvajanja programa preverjali obstoj omenjene nevarnosti, bi lahko prišel na vrsto prepozno. Zaradi tega je v tem primeru treba ukrepati s prekinitvijo izvajanja programa takoj ob pojavu signalizacije kritičnega dogodka ter izvršiti pripadajočo programsko sekvenco (slika 4. 7). Šele po tem ukrepu se potek izvajanja programa nadaljuje v točki, kjer je prišlo do prekinitve.



Slika 4. 7: Prekinitve izvajanja programa

Načeloma obstajata dve vrsti prekinitrov z ozirom na izvor zahteve po prekinitvi: softverska in hardverska (glej tudi tabelo 4. 2)<sup>11</sup>. V prvem primeru imamo opravka s prekinitvami, ki so običajno posledica napak ali nelogičnosti v programu (npr. deljenje z 0, nepravilen ukaz, ukaza CHK in CHK2 itd.). Hardverske prekinitve so posledica reakcije na zunanji signal, ki ga pripeljemo na ustrezno definirane binarne vhode (RESET, IRQ7 do IRQ1; glej blokovno shemo na sliki 3.3). Prekinitve izvajanja programa v CPU lahko zahtevajo tudi drugi podmoduli MC68332 (npr. TPU po predhodno določenem številu preštetih pulzov na vhodnem kanalu ali QSPI pri serijskem prenosu).

V grobem je zaporedje dogodkov pri prekinitvah naslednje:

<sup>10</sup> Uporabno npr. pri baterijskem napajanju. S tem preide mikrokrumilnik v stanje, ko se mu aktivnost in poraba zmanjšata na minimum, potreben za ohranjanje osnovnih informacij. Stanje traja do ponovnega "prebujanja" ob določeni prekinitvi.

<sup>11</sup> V Motorolinem izrazoslovju označujejo splošno prekinitve ne glede na njen vzrok z izrazom "exception" (izjema), hardverske prekinitve pa dodatno imenujejo "interrupts" ("prekinitve"). V običajnem žargonu uporabljam izraza "softverska in hardverska prekinitve".

1. Zahteva po prekinitvi: zahtevnik prekinitve "zaprosi" CPU za prekinitve izvajanja programa.
2. CPU zazna zahtevnika prekinitve in mu ustreže (glej podpoglavlje 4.2.8.1.1) ali pa ga ignorira.
3. Izvajanje se nadaljuje na prekinitvenem podprogramu, ki se obvezno konča z ukazom RTE (glej podpoglavlje 4.2.8.1.2)!
4. Izvajanje programa se nadaljuje z ukazom v "glavnem programu", ki je bil na vrsti, preden je prišlo do zahteve po prekinitvi.

#### 4.2.8.1.1 Hierarhija in servisiranje prekinitiv

Nekatere prekinitve se izvajajo brezpogojno, druge pa lahko onemogočimo. Primer brezpogojne prekinitve je hardverska prekinitve RESET, ki reinicializira celoten operacijski sistem: izvajanje programa se dokončno prekine in vsi registri se postavijo v začetno stanje. Prekinitve so namreč glede na prednost (prioritet) razdeljene na več skupin. Tako ima hardverska prekinitve IRQ7 najvišjo, IRQ1 pa najnižjo prioriteto. Vse ostale prekinitve razen IRQ7 lahko onemogočimo s t.i. "*maskiranjem*". Izraz pomeni postavitev področja I2, I1, I0 (prekinitvena maska) v sistemskem zlogu SR v kombinaciji, ki dovoljuje le nekatere prekinitve (podpoglavlje 4.1.2). Omogočene so vse prekinitve, katerih zaporedna številka je večja od binarnega števila v prekinitveni maski. Če se v prekinitveni maski nahaja kombinacija  $100_{BIN} = 4_{DEC}$ , sta IRQ5 in IRQ6 omogočeni, IRQ1 - IRQ4 pa ne. Edina izjema je IRQ7, ki je tudi kombinacija  $111_{BIN} = 7_{DEC}$  ne more preprečiti. Hierarhija prekinitiv pride zlasti do izraza v primeru hkratne zahteve po več prekinitvah. V takšnem primeru imajo prekinitve z višjo prioriteto prednost pri izvajanju.

V fazi inicializacije moramo tudi določiti začetni naslov podprograma, ki se bo izvajal ob določeni prekinitvi. V t.i. "vektorski tabeli" se nahajajo kazalci na začetne lokacije podprogramov vseh prekinitrov (t.i. vektorji). Programer mora v vektorje predvidenih prekinitrov vpisati začetne naslove ustreznih prekinitvenih podprogramov. Tabela 4. 2 kaže vektorje in njihove naslove v vektorski tabeli za nekatere prekinitve

| Številka vektorja | Odmik vektorja |            | Opis   |
|-------------------|----------------|------------|--|
|                   | DEC            | HEX        |  |
| 0                 | 0              | 0          | Reset: začetni SP                                  |
| 1                 | 4              | 4          | Reset: začetni PC                                  |
| 3                 | 12             | c          | Napaka pri naslavljjanju                           |
| 4                 | 16             | 10         | Nelegalni ukaz                                     |
| 5                 | 20             | 14         | Deljenje z 0                                       |
| 48-58             | 192<br>232     | c0<br>e8   | Rezervirano za koprocessor                         |
| 64-255            | 256<br>1020    | 100<br>3fc | Vektorji, ki jih definira uporabnik (npr. za TPU). |

**Tabela 4. 2: Vektorska tabela**

Položaj vektorske tabele v pomnilniku (njen začetek) določa vsebina registra VBR (glej podpoglavlje 4.1.2). Na ta način lahko s spremembo vsebine VBR določimo več vektorskih tabel.

### Primer:

Če hočemo napisati lasten podprogram, ki se bo izvajal v primeru operacije deljenja z nič, moramo med inicializacijo na lokacijo (VBR) + 14<sub>HEX</sub> shraniti naslov začetnega ukaza tega prekinitvenega podprograma.

#### 4.2.8.1.2 Vrnitev iz prekinitvenega podprograma

Ob zahtevi po prekinitvi se vsebina SR shrani v sklad. Po skoku na prekinitveni podprogram se v sklad shrani tudi vsebina PC, enako kot pri skokih na navadne podprograme. Ta "stari" PC kaže na ukaz, pred katerim je prišlo do prekinitve. Temu ustrezno se spremeni tudi SP. To stanje je treba po opravljeni prekinitveni proceduri restavrirati, za kar poskrbi ukaz RTE, podobno kot RTS pri vrnitvi iz navadnih podprogramov.

## 4.3 Načini naslavljjanja

Večina ukazov ima na voljo več možnosti definiranja izvora in končnega cilja operandov. Izbira primerenega načina naslavljjanja je zelo pomembna, saj neposredno vpliva na hitrost izvajanja in obsežnost programa. V tem podpoglavlju si bomo ogledali le najpogosteje načine naslavljjanja, ki so skupni večini mikroprocesorjev, podrobnejše informacije o CPU32 pa so v literaturi [25]. Pri ukazih z dvema operandoma lahko načeloma za vsak operand uporabimo drugačen načina naslavljjanja.

### 4.3.1 Neposredno naslavljjanje registrov

#### Oznaka: Dn ali An

Pri tem načinu naslavljjanja (angl. direct addressing) se operandi nahajajo v podatkovnem ali naslovнем registru.

Primer:

```
add.l A0,D7      *vsebina A0 + vsebina D7 => D7
move.b D3,D4      *vsebina byta z najnižjo težo iz D3 => D4
```

Značilnost tega naslavljjanja je, da se eden (ali oba) operanda nahajata v registrih CPU. Na ta način dosežemo najhitrejše izvajanje operacij, saj kakršnokoli drugačno naslavljjanje, ki operira s pomnilnikom (posebej, če le-ta ni na čipu), zahteva več urinih ciklov.

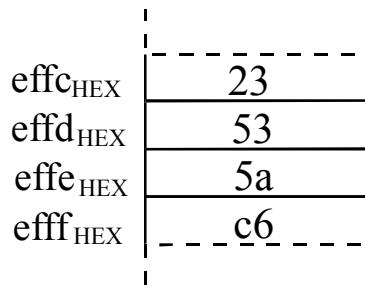
### 4.3.2 Posredno naslavljjanje s pomočjo naslovnega registra

**Oznaka:** (An)

Pri tem načinu naslavljanja (angl. address register indirect) je eden od operandov (ali oba) shranjen v pomnilniku na naslovu, ki ga določa naslovni register.

Primer:

```
MOVE.L (A0),D2    *□e je vsebina A0 effcHEX, vsebina te spominske
                  *lokacije pa takšna kot na sliki 4. Če bo
                  *vsebina registra D2 po ukazu 23535ac6HEX
```



Slika 4. 8: Primer vsebine spominske lokacije

### 4.3.3 Posredno naslavljjanje s pomočjo naslovnega registra z naknadnim inkrementiranjem

**Oznaka:** (An)+

Ta način je podoben prejšnjemu, le da se vsebina naslovnega registra (naslov operanda) po ukazu samodejno poveča in kaže na naslednjo lokacijo. Povečanje je odvisno od uporabljenega tipa operanda (npr. pri "long" operandih so to štirje byti).

Primer:

```
MOVE.L (A0)+,D2    *□e je vsebina registra A0 effcHEX, vsebina
                  *te spominske lokacije pa takšna kot na
                  *sliki 4. Če bo po ukazu vsebina D2
                  *23535ac6HEX, vsebina A0 pa f000HEX
```

Način naslavljanja je zelo uporaben pri dostopu do zapovrstnih podatkov, npr. elementov tabele.

#### 4.3.4 Posredno naslavljjanje s pomočjo naslovnega registra ob predhodnem dekrementiranju

**Oznaka:** -(An)

Zopet imamo opravka z naslavljjanjem, ki je podobno predhodnima dvema. Razlika je le v tem, da se pred uporabo naslovnega registra njegova vsebina zmanjša za 1, 2 ali 4. Dekrement je odvisen od podatkovne širine operanda v ukazu. Pri bytnem operandu je enak ena, pri besednjem dve, pri dolgi besedi pa štiri.

Primer:

```
MOVE.L -(A0),D2      *□e je vsebina A0 f000HEX, vsebina te  
                      *spominske lokacije pa kot na sliki 4. □,  
                      *se pred uporabo v ukazu vsebina A0  
                      *spremeni v effHEX, vsebina D2 pa bo  
                      *potem takem 23535ac9HEX.
```

#### 4.3.5 Takošnje podajanje podatka

**Oznaka:** #XXX

Pri določenih ukazih (s podaljškom "I" - angl. Immediate - takojšen) lahko enega izmed operandov podamo neposredno v samem ukazu.

Primer:

```
addi.b #$23,D0      *k vsebini spodnjega byta registra D0  
                      *(stara vsebina f0HEX) prištej 23HEX.  
                      *Rezultat: (D0)=13HEX. PAZI! Zaradi bytnega  
                      *operanda se presežek ne prenese na višji  
                      *byte.
```

#### 4.3.6 Absolutni dolgi naslov

**Oznaka:** (XXX).L

S tem načinom (angl. absolute long address) neposredno podajamo naslov, na katerem se nahaja operand.

Primer:

```
NEG.L ($123456).L    *negiranje vsebine spominske lokacije z
```

\*naslovom 123456<sub>HEX</sub>.  
CLR.W (&8000).L      \*brisanje besede na naslovu  
\*8000<sub>DEC</sub> = 1f40<sub>HEX</sub>.

#### 4.3.7 Posredno naslavljjanje z indeksom (in osnovnim odmikom)

**Oznaka:** (bd, An, Xn, SIZE\*SCALE)

Oglejmo si sedaj primer nekoliko bolj zapletenih načinov naslavljanja. Naslov, na katerem se nahaja podatek, se računa tako, da naslovu v An prištejemo vrednost osnovnega odmika (angl. basic displacement - bd) ter skalirano vrednost indeksnega registra:

$$ea = bd + (An) + (Xn * SCALE).$$

Kot indeksni register lahko uporabljamo podatkovni ali naslovni register. V njem lahko definiramo 16-bitni (sufiks W) ali 32-bitni indeks (sufiks L). Vrednost indeksnega registra lahko množimo s faktorjem SCALE, katerega možne vrednosti so 1, 2, 4 ali 8.

Kot primer vzemimo ukaz: MOVE.W D0,(8000,A0,D1.L\*2).

Pred ukazom so vrednosti posameznih registrov:

$$\begin{aligned} (d0) &= 1234_{HEX}, \\ (d1) &= 1000_{HEX}, \\ (a0) &= 2000_{HEX}. \end{aligned}$$

Efektivni naslov, na katerega shranimo besedo iz D0, izračunamo po zgornji enačbi:

$$8000_{HEX} + 2000_{HEX} + 1000_{HEX} \cdot 2 = b000_{HEX},$$

torej bo po izvršitvi ukaza:

$$(b000) = 1234_{HEX}$$

## 4.4 Čas izvajanja posameznega ukaza

Čas, ki je potreben za izvajanje posameznega ukaza, je vsekakor eden od pomembnih parametrov pri presojanju zmogljivosti procesorja. Pri nekaterih procesorjih (npr. pri digitalnih signalnih procesorjih - DSP) je ta čas ob enakem načinu naslavljanja enak za vse ukaze<sup>12</sup>, razen pri operaciji deljenja, kjer traja nekajkrat več (npr. 16-krat pri 16-bitnem deljenju).

Pri običajnih procesorjih, kamor sodi tudi CPU32, je izračun trajanja določene programske sekvence zapleten, saj je čas izvajanja posameznega ukaza odvisen od naslednjih faktorjev:

---

<sup>12</sup> Npr. pri TMS320F240 je to 50 ns pri najhitrejšem načinu naslavljanja.

- tipa ukaza,
- načina naslavljanja,
- formata operandov,
- urinega takta, saj lahko isti procesor dela na različnih frekvencah.

Izračun potrebnih časovnih ciklov je sestavljen iz več segmentov<sup>13</sup>. V naslednji tabeli so za ilustracijo prikazani faktorji za izračun urinih ciklov, ki so potrebni za izvajanje nekaterih značilnih ukazov. Samega algoritma izračuna in dodatnih ciklov, ki so odvisni od načina naslavljanja in formata operandov, ne bomo navajali. Že sama razmerja med različnimi ukazi pa ilustrirajo velike razlike v trajanju izvajanja.

| Ukaz                | Začetni cikli<br>(head) | Končni<br>cikli<br>(tail) | Cikli     |
|---------------------|-------------------------|---------------------------|-----------|
| ADD Rn, Rm          | 0                       | 0                         | 2(0/1/0)  |
| OR <FEA>, Dn        | 0                       | 0                         | 2(0/1/0)  |
| MOVE Rn, Rn         | 0                       | 0                         | 2(0/1/0)  |
| MOVE.L <FEA>, (An)  | 2                       | 2                         | 6(0/1/2)  |
| ADDI #, Rn          | 0                       | 0                         | 2(0/1/0)  |
| ADDI #, <FEA>       | 0                       | 3                         | 5(0/1/2)  |
| BSET Dn, Dm         | 4                       | 0                         | 6(0/1/0)  |
| RTE                 | 1                       | -2                        | 24(4/2/0) |
| MULS(U).W <FEA>, Dn | 0                       | 0                         | 26(0/1/0) |
| DIVS.W <FEA>, Dn    | 0                       | 0                         | 2(0/1/0)  |

**Slika 4. 9: Tabela za izračun časovnih ciklov potrebnih za izvajanje nekaterih ukazov CPU32**

Iz tabele je razvidno, da so razlike med časi izvajanja posameznih ukazov zelo velike, še zlasti pri aritmetičnih operacijah seštevanja, odštevanja, množenja in deljenja.

Izkušen programer se bo v časovno kritičnih programih skušal izogniti uporabi časovno zahtevnejših ukazov (kot so npr. deljenja in množenja). Oglejmo si to na primeru množenja spremenljivke, ki se nahaja na naslovu 8000<sub>HEX</sub>, s faktorjem 7.

V prvem trenutku se najenostavnejša zdi uporaba ukaza za množenje:

```

LEA    $8000,a5  * naslov spremenljivke v a5
MOVE.W (a5),d1  * nalaganje spremenljivke v d1
MULS.W #7,d1    * množenje s 7
MOVE.W d1,(a5)  * shranjevanje rezultata

```

Naslednja rešitev je nekoliko hitrejša, čeprav rabimo več ukazov:

<sup>13</sup>Število ciklov za izvajanje instrukcije, cikli za zaključek predhodne instrukcije, cikli za bralni in vpisovalni pristop, zajemanje instrukcije (angl. fetch) itd.

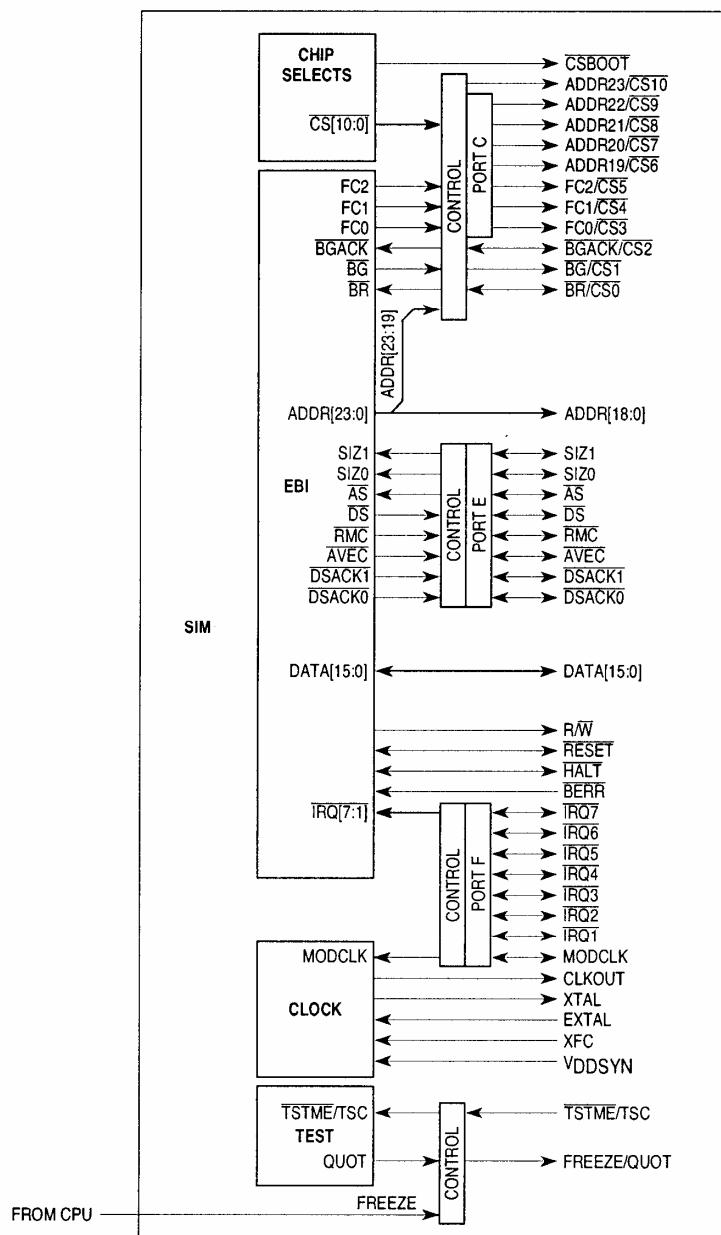
```
LEA    $8000,a5 * naslov spremenljivke v a5
MOVE.W (a5),d1 * nalaganje spremenljivke v D1
MOVE.W d1,d0 * preslikava spremenljivke v D0
LSL.W #3,d0 * pomik vsebine D0 za 3 mesta (bite) v
              levo
              * (množenje z 8)
SUB.W d1,d0 * odštevanje (D0) od (D1): D0=(8 - 1)*D1
MOVE.W d0,(a5) * shranjevanje rezultata
```

Zgornjo rešitev bi dobili, če bi množenje s konstanto izvedli v višjem jeziku (npr. C) ob časovni optimizaciji.

## 5. MODUL ZA INTEGRACIJO SISTEMA - SIM

SIM (angl. System Integration Module - modul za integracijo sistema) je modul mikrokrmlnika MC68332, katerega glavna naloge so [24]:

- povezovanje s periferijo,
- generiranje urinega takta za ostale module,
- sistemski zaščita,
- krmiljenje in zajemanje vhodno/izhodnih digitalnih signalov
- generiranje signalov za izbiro čipov.



Slika 5. 1: Blokovna shema SIM z nožicami

Funkcionalni bloki SIM so (slika 5. 1):

- **konfiguracija sistema in zaščita** (angl. System Configuration and Protection),
- **sintetizator ure** (angl. Clock Synthesizer),
- **preizkušanje sistema** (angl. System Test),
- **vmesnik k zunanjem vodilu** (angl. External Bus Interface - EBI),
- **izbira integriranih vezij - čipov** (angl. chip selects - CS).

Večina nožic SIM je večnamenskih (npr. nožico IRQ1 lahko uporabljamо kot vhod za hardversko prekinitve ali pa kot navadni binarni vhod ali izhod). Pred zagonom programa je za izbiro funkcije in nastavljanje parametrov SIM treba v ustreznа stanja postaviti (“maskirati”) bite na vnaprej določenih spominskih lokacijah. Čeprav gre v tem primeru za RAM pomnilnik in ne za prave registre (kot so npr. podatkovni in naslovni registri CPU), imenujemo te 16-bitne lokacije “SIM registri”. Njihove naslove ter nazive kaže slika 5. 2<sup>1</sup>.

|        | 15     | 8 7                                  | 0 |
|--------|--------|--------------------------------------|---|
| Yffa00 |        | MODULE CONFIGURATION (MCR)           |   |
| Yffa02 |        | MODULE TEST (SIMTR)                  |   |
| Yffa04 |        | CLOCK SYNTHESIZER CONTROL (SYNCR)    |   |
| Yffa06 | PRAZNO | RESET STATUS (RSR)                   |   |
| Yffa08 |        | MODUL TEST E (SIMTRE)                |   |
| ...    |        | PRAZNO                               |   |
| Yffa10 | PRAZNO | PORTE DATA (PORTE)                   |   |
| Yffa12 | PRAZNO | PORTE DATA (PORTE)                   |   |
| Yffa14 | PRAZNO | PORTE DATA<br>DIRECTION (DDRE)       |   |
| Yffa16 | PRAZNO | PORTE PIN<br>ASIGNMENT (PEPAR)       |   |
| ...    |        | enako za PORTF                       |   |
| Yffa20 | PRAZNO | SYSTEM PROTECTION<br>CONTROL (SYPCR) |   |
| Yffa22 |        | PERIODIC INTERRUPT CONTROL (PICR)    |   |
| Yffa24 |        | PERIODIC INTERRUPT TIMING (PITR)     |   |
| Yffa26 | PRAZNO |                                      |   |
|        | PRAZNO | SOFTWARE SERVICE<br>(SWSR)           |   |
| Yffa28 |        | PRAZNO                               |   |
| ...    |        | ....                                 |   |
| Yffa44 |        | CHIP SELECT PIN ASSIGNMENT (CSPAR0)  |   |
| Yffa46 |        | CHIP SELECT PIN ASSIGNMENT (CSPAR1)  |   |
| Yffa48 |        | CHIP SELECT BASE BOOT (CSBARBT)      |   |
| Yffa4a |        | CHIP SELECT OPTION BOOT (CSORBT)     |   |
| ...    |        | ... registri ostalih <u>CS</u> ...   |   |
| Yffa58 |        | CHIP SELECT BASE 3 (CSBAR3)          |   |
| Yffa5a |        | CHIP SELECT OPTION 3 (CSOR3)         |   |
| ...    |        | ... registri ostalih <u>CS</u> ...   |   |
| Yffa4a |        | PRAZNO                               |   |

*Slika 5. 2: RAM lokacije in nazivi SIM registrov*

<sup>1</sup> Prva heksadecimalna števka Y v naslovu je lahko 7<sub>HEX</sub> ali F<sub>HEX</sub>, odvisno od bita MM v MCR registru.

Nimamo namena podrobno opisati posamezne registre. Omejili se bomo le na nekatere značilnosti SIM.

## 5.1 Podmodul za konfiguracijo sistema

Podmodul za **konfiguracijo sistema in zaščito** skrbi za osnovno prednastavitev SIM. V ta namen v registru za konfiguracijo modula - MCR (angl. Module Configuration Register) na primer določimo, ali bo prva števka iz registrske mape SIM 7<sub>HEX</sub> ali F<sub>HEX</sub>. Posebno polje določa prioriteto prekinitve SIM modula v primerjavi z zahtevami po prekinitvah ostalih modulov MC68332 itd.

V SYPCR registru (angl. System Protection Control Register) lahko omogočimo in parametriramo čuvaju časa (angl. *watchdog* - dobesedno: pes čuvaj). Osnovna funkcija watchdoga je nadzor nad trajanjem izvajanja programa. Namreč, velika večina programov za delo v realnem času (npr. regulacija motorjev), se izvaja ciklično (običajno kot posledica časovne prekinitve, glej pogl. 10). Celotna programska sekvenca se mora končati preden poteče nek vzorčni interval  $T_V$  (glej tudi [2]). Eden od možnih vzrokov za "podaljšanje" izvajanja te sekvence je neskončna zanka, ki je posledica nepravilno zasnovanih internih skokov. Zato je koristno predvideti nadzor nad časom izvajanja programa.

Watchdog poženemo v SYPCR registru, kjer tudi določimo trajanje nadzornega cikla (od 30,6 µs do 1 s pri 16,77 MHz osnovnem taktu). Po zagonu watchdoga je treba register SWSR (angl. Software Service Register) periodično (običajno na koncu cikla uporabniškega programa) postaviti v določeno stanje, ki sproži naslednje odštevanje časa. Izostanek te procedure pomeni, da je izvajanje programa daljše od časa, predvidenega v watchdog časovniku, nakar ta povzroči RESET prekinitve.

Poleg omenjene funkcije lahko v podmodulu za konfiguracijo sprožimo časovnik za periodično prekinitve (angl. periodic interrupt timer). Posledica je skok v prekinitveni podprogram v pravilnih časovnih intervalih, zato lahko to funkcijo uporabljam za zagotavljanje intervala vzorčenja. Pomanjkljivost tega pristopa je v relativno grobi izbiri možnega intervala med dvema prekinitvama (od 122 µs do 15,9 z minimalnim intervalom 122 µs pri 16,77 MHz urinem taktu; npr. prekinitve s periodo 1 ms ni mogoče realizirati). Zato za generiranje časovne prekinitve raje uporabljam PWM funkcijo TPU (pogl. 7).

## 5.2 Sintetizator ure

Frekvence sistemске ure za MC68332 so (odvisno od verzije) 16,77 MHz, 20,97 MHz ali 25 MHz (glej poglavje 4 o CPU). *Kristal*, dajalnik referenčnega takta za uro, pulzira s bistveno nižjo frekvenco; pri prvi verziji se lahko giblje med 25 kHz in 50 kHz (nazivna vrednost 32,768 kHz)! **Sintetizator ure** skrbi za nastavljanje sistemski frekvence prek maskiranja ustreznih bitov v registru SYNCR (angl. Clock Synthesizer Control Register). Na ta način je možna fleksibilna softverska izbira sistemski frekvence, ki daje osnovni takt vsem operacijam mikrokrmilnika! Pri frekvenci kristala 32,768 kHz je omogočena nastavitev sistemski ure prve inačice od 131 kHz do 16,777 MHz. Referenčni takt pa lahko zajemamo tudi iz zunanjega vira.

## 5.3 Preizkušanje sistema

Namen podmodula za **preizkušanje sistema** je tovarniško testiranje delovanja mikrokrmlnika, zato je uporabniku nedostopen.

## 5.4 Vmesnik k zunanjem vodilu

### 5.4.1 Povezava s perifernim vodilom

Na sliki 5. 1 so prikazane vhodno/izhodne nožice vmesnika k zunanjem vodilu. Lahko jih razdelimo v dve osnovni skupini: **zunanje naslovno, krmilno in podatkovno vodilo** s **hardverskimi prekinivami** ter **nožice za izbiro integriranih vezij (čipov)**. Slednje bomo obravnavali v posebnem podpoglavlju.

Zunanje naslovno, podatkovno in krmilno vodilo rabimo za naslavljjanje zunanjih enot (vezij) kot so pomnilniki, A/D in D/A pretvorniki in ostala vezja. Zaradi pomembnosti bomo nožice za hardverske prekinitve obravnavali ločeno.

Celotno **naslovno vodilo** MC68332 obsega 24 nožic (A0-A23), kar pomeni možnost naslavljanja  $2^{24} = 16\ 777\ 216$  zlogov (16 Mbyte<sup>2</sup>) perifernih enot. V tem segmentu je zajetih le prvih 19 naslovnih bitov (A0-A18), preostalih pet (A19-A23) pa sodi k modulu za izbiro čipov (glej podpoglavlje 5.5).

Interni **podatkovno vodilo** (IMB) omogoča hkratni transfer 32-bitnega podatka med podmoduli v enem samem ciklu. Po drugi strani obsega zunanje vodilo le 16 bitov (D0 - D15). To pomeni, da moramo 32-bitni podatek v/iz periferije poslati/sprejeti v dveh obrokih. Hitrost prenosa podatkov v ali iz zunanjih pomnilnikov je manjša od hitrosti internega prenosa znotraj mikroracunalnika. Obe dejstvi sta razlog za to, da se izkušeni programerji trudijo čim manj komunicirati z zunanjimi pomnilniki. Prav v ta namen poskušajo projektanti procesorjev zagotoviti veliko število internih podatkovnih registrov (pri nas jih imamo osem: D0 - D7). Če vmesnega rezultata neke operacije ne rabimo takoj po njenem izvajanju, ampak šele čez nekaj ukazov, raje ta podatek ohranimo v registru iz katerega ga po potrebi pokličemo. Seveda zahteva takšen pristop premišljeno pisanje programa, ker je tak register do nadaljnjega zaseden in neuporaben za druge operacije<sup>3</sup>.

Prilagoditev hitrosti transferov med hitrim CPU in počasnejšim pomnilnikom dosežemo z uvajanjem *čakalnih stanj* (angl. wait states).

**Krmilno vodilo** (angl. control bus) je skorajda v celoti zbrano v *vratih* (angl. port) E (slika 5. 1). Ta imajo dvojno vlogo: njihove nožice lahko delujejo v funkciji krmilnega vodila ali pa kot navadnih osem binarnih vhodov ali izhodov. Za izbiro funkcije moramo maskirati registre PEPEAR, DDRE in PORTE (slika 5. 2).

---

<sup>2</sup> PAZI! 1 Kbyte ni 1000, temveč 1024 bajtov.

<sup>3</sup> Med ostalim je tudi to prednost uporabe strojnega jezika pred C-jem, saj slednji, zlasti pri skokih na podprograme, shranjuje vmesne podatke v RAM.

Kot primer lahko omenimo delovanje izhodov SIZ1 in SIZ0 ter vhoda DSACK0 in DSACK1, ki skupaj z bitom naslovnega vodila A0 sinhronizirajo komunikacijo s periferijo. Pogosto se namreč dogaja, da je nek podatek treba poslati ali prečitati v/z 8- ali 16-bitne periferne enote. Ena od nalog omenjenih signalov je prilagoditev 32-bitnega ali 8-bitnega podatka iz CPU na enoto z drugačno dolžino vodila, enako kot izbira pozicije poslanega byta (soda ali liha). Pri tem pa izhoda AS in DS (angl. address strobe, data strobe) signalizirata aktivnost naslovnega oz. podatkovnega vodila.

#### 5.4.2 Digitalne vhodno/izhodne nožice

Ena od glavnih značilnosti mikrokrmlnika MC68332 je možnost izbire funkcije nožic. Večina nožic v SIM in QSM ima vsaj dvojno funkcijo; poleg specifičnih funkcij (vodila, CS nožice, hardverske prekinitve...) jih lahko uporabljamo kot splošne digitalne vhodno/izhodne nožice. Ti priključki so zbrani v vrath E, F in C (z določenimi omejitvami). Po izbiri funkcije nožice (PFPAR in PEPAR: 1 - standardna funkcija s slike 5. 1, 0 - vhodno/izhodna funkcija) izberemo smer digitalnih signalov (0 - izhod, 1 - vhod) v registrih DDRE in DDRF. V registrih PORTE0 in PORTE1 in PORTF0 in PORTF1 določimo vrednost izhodnega signala ali preberemo vrednost vhodnega signala (slika 5. 2).

#### 5.4.3 Hardverske prekinitve

Sedem nožic na portu F lahko uporabljamo kot vhode za zunanje hardverske prekinitve (IRQ1-IRQ7; angl. interrupt request - zahteva po prekinitvi). Vhodi imajo že vnaprej določeno prioriteto: ob hkratni zahtevi po več zunanjih prekinitvah ima IRQ7 največjo, IRQ1 pa najmanjšo prioriteto. Vse prekinitve razen IRQ7 lahko maskiramo v statusnem registru CPU (glej tudi pogl. 4.2.8.1.1). Če je prekinitvev odobrena, pride do izvajanja prekinitvenega programa za ustrezeni IRQ. Naslovi prekinitvenih podprogramov se nahajajo v vektorski tabeli na lokacijah  $64_{HEX}$  -  $7c_{HEX}$ .

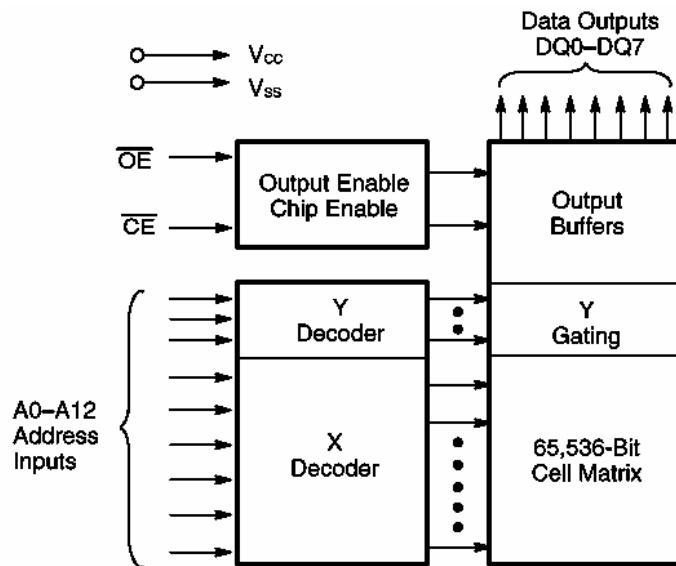
### 5.5 Izbira integriranih vezij

#### 5.5.1 Problematika naslavljjanja čipov

24-bitno naslovno vodilo omogoča naslavljjanje 16 M ( $000000_{HEX}$  -  $FFFFF_{HEX}$ ) bytnih lokacij. Znotraj tega intervala so nekatera področja rezervirana za različne registre podmodulov ali pa za inicializacijski program ter delovni RAM (glej pogl. 8). Ostala nerezervirana področja lahko uporabimo za naslavljjanje zunanjih enot (RAM in ROM pomnilnikov, pretvornikov, prikazovalnikov itd.). Pri snovanju arhitekture našega mikroračunalniškega sistema moramo seveda poskrbeti za to, da so naslovi zunanjih enot enoumno določeni, torej dve lokaciji ne smeta imeti istega naslova.

Oglejmo si primer naslavljjanja 8 Kbyte RAM pomnilnika ( $8192_{DEC} = 2000_{HEX}$  lokacij velikosti enega byta), ki je prikazan na sliki 5. 3. Pomnilnik želimo postaviti na prosto področje  $F00000_{HEX}$  -  $F01FFF_{HEX}$ . Na vezju so za to zadolžene naslednje nožice (podobno velja tudi za ostala vezja):

- naslovne nožice za priključitev na naslovno vodilo mikročunalnika (A12 - A0),
- nožica za izbiro vezja CS.



**Slika 5. 3: Blokovna shema pomnilnika AMD27X64 (Advanced Micro Devices) velikosti 8 Kbyte x 8-bit**

Navedeni interval naslovov pomnilnika lahko zapišemo tudi v binarni obliki

1111 0000 0000 0000 000<sub>BIN</sub> - 1111 0000 0001 1111 1111 1111<sub>BIN</sub>,

kjer se vsako naslavljjanje kombinacije

1111 0000 000X XXXX XXXX XXXX<sub>BIN</sub><sup>4</sup>

nanaša na neko lokacijo znotraj izbranega RAM. Skrajnje levi bit (MSB) ustreza naslovnemu bitu A23, skrajnji desni (LSB) pa A0.

Iz opisanega lahko ugotovimo naslednje:

- Biti, označeni z X, določajo naslove v intervalu 8 K znotraj čipa (lokalni naslovi).
- Preostala kombinacija bitov določa absolutno lokacijo čipa, to je položaj teh 8K v spominski mapi MC68332.

Ustrezno temu sledijo praktični napotki:

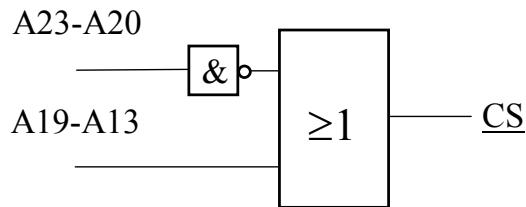
- X bite moramo vezati neposredno na naslovne nožice vezja A12 - A0 ( $2^{13} = 8192_{DEC} = 8\text{ K}$ ).

---

<sup>4</sup> X pomeni logično stanje 0 ali 1.

2. Čip bo naslovljen le, če se bo na naslovnih nožicah A23 - A13 mikrokrumilnika pojavila kombinacija 1111 0000 000, torej je treba omogočiti, da se ob njej aktivira signal CS (aktivien na logično 0).

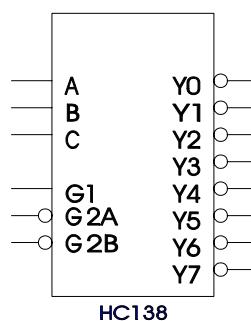
Najprimitivnejše vezje za izbiro čipa je izrisano na sliki 5. 4.



**Slika 5. 4: Principialna logična shema vezja za omogočanje CS**

Za realizacijo vezja s prejšnje slike bi rabili nekaj čipov: naslovne bite A23 - A20 je treba povezati preko logične funkcije NEIN (angl. NAND), njen izhod pa skupaj z ostalimi biti A19 - A13 pripeljemo na logično funkcijo ALI (angl. OR). Takrat se le ob izbrani kombinaciji bitov na izhodu pojavi logična 0. Za realizacijo opisanega pristopa bi potrebovali precej čipov. Če temu dodamo še dejstvo, da je treba podobna vezja realizirati tudi za izbiro ostalih perifernih integriranih vezij, kmalu ugotovimo, da bi rabili ogromno čipov z logičnimi funkcijami, kar bi pomenilo veliko "gnečo" na tiskanem vezju.

Omenimo še dodatni problem naslavljanja čipov, ki zasedeta spominsko področje le enega ali dveh bytov. Takšni so npr. A/D in D/A pretvorniki (imajo tudi CS vhod). 12-bitni pretvornik moramo nasloviti kot psevdospominsko lokacijo (v tem primeru sta to 2 byta; štirje biti so nezasedeni). Če ga želimo nasloviti s 24-bitnim naslovnim vodilom, moramo prek raznih logičnih funkcij povezati 23 bitov (A0 uporabimo za izbiro med enim ali drugim bytom)! Naslavljanje nekaj takšnih vezij na opisani način bi zahtevalo izredno veliko logičnih vezij, zato ga srečamo le poredkom. Za izbiro CS raje uporabljam dekoderje (sliki 5. 5 in 5. 6).



**Slika 5. 5 Blokovna shema dekoderja SN74F138**

Dekoder s slike lahko združi tri naslovne bite. Vsaka od osmih ( $2^3$ ) kombinacij bo postavila le enega od izhodov v stanje 0, vsi ostali pa bodo setirani. Čeprav smo z njihovo uporabo bistveno zmanjšali potrebno število logičnih elementov, je očitno, da bo "široko" naslovno vodilo še vedno zahtevalo veliko dekoderjev.

| Sprostitveni vhodi |     |     | Izbirni vhodi |   |   | Izhodi |    |    |    |    |    |    |    |
|--------------------|-----|-----|---------------|---|---|--------|----|----|----|----|----|----|----|
| G1                 | G2A | G2B | C             | B | A | Y0     | Y1 | Y2 | Y3 | Y4 | Y5 | Y6 | Y7 |
| x                  | 1   | x   | x             | x | x | 1      | 1  | 1  | 1  | 1  | 1  | 1  | 1  |
| x                  | x   | 1   | x             | x | x | 1      | 1  | 1  | 1  | 1  | 1  | 1  | 1  |
| 0                  | x   | x   | x             | x | x | 1      | 1  | 1  | 1  | 1  | 1  | 1  | 1  |
| 1                  | 0   | 0   | 0             | 0 | 0 | 0      | 1  | 1  | 1  | 1  | 1  | 1  | 1  |
| 1                  | 0   | 0   | 0             | 0 | 1 | 1      | 0  | 1  | 1  | 1  | 1  | 1  | 1  |
| 1                  | 0   | 0   | 0             | 1 | 0 | 1      | 1  | 0  | 1  | 1  | 1  | 1  | 1  |
| 1                  | 0   | 0   | 0             | 1 | 1 | 1      | 1  | 1  | 0  | 1  | 1  | 1  | 1  |
| 1                  | 0   | 0   | 1             | 0 | 0 | 1      | 1  | 1  | 1  | 0  | 1  | 1  | 1  |
| 1                  | 0   | 0   | 1             | 1 | 0 | 1      | 1  | 1  | 1  | 1  | 1  | 0  | 1  |
| 1                  | 0   | 0   | 1             | 1 | 1 | 1      | 1  | 1  | 1  | 1  | 1  | 1  | 1  |

**Slika 5. 6: Izjavnostna tabela dekoderja SN74F138**

Opisani problem je možno elegantno rešiti s pomočjo PAL vezij (angl. Programmable Array Logic). To so čipi, v katerih lahko "programiramo" tudi zelo zapletene logične povezave vhodov in izhodov s postopkom, ki je podoben programiranju EPROM pomnilnikov.

### 5.5.2 Izbira čipov v MC68332

Pojektanti mikrokrmilnikov serije MC68300 so se zavedali opisanega problema. V ta namen so vpeljali programsko definirane CS izhode.

V ustreznih registrih SIM modula lahko določimo naslove za aktiviranje ene izmed dvanajstih mikrokrmilnikovih CS nožic (CSBOOT ter CS10 - CS0), ki bo povezana z vhodom CS perifernega čipa.

CSBOOT je rezervirana za t.i. *začetno nalaganje* (tudi začetni zagon, angl. bootstrap). V našem primeru pomeni to nalaganje inicializacijskega programa iz ROM (običajno EPROM) pomnilnika ob zagonu oz. resetiranju (reset prekiniti ob aktiviranju RESET nožice). Zato mora imeti vsak dobro zasnovan sistem z MC68332 predviden že programirani pomnilnik, katerega CS je vezan na CSBOOT.

Preostale nožice (CS10 - CS0) so načeloma na voljo uporabniku.

#### **OPOMBA!**

V BCC izvedbi mikroračunalniškega sistema s PFB (pogl. 8) so vse CS nožice, **razen CS3** pri "B" verziji, že uporabljeni za naslavljanje določenih čipov (npr. CPU32Bug EPROM, RAM, eventualnega matematičnega koprocesorja...).

Bistvo uporabe CS nožic v MC68332 je v možnosti softverske izbire naslova za njihovo aktiviranje. Za to sta zadolžena dva registra (slika 5. 7): CSBARx in CSORx (x ustreza zaporedni številki CS; glej sliko 5. 2).

Pri definiranju pomena posameznih bitov v registrih vseh podmodulov (SIM, QSM in TPU) veljajo naslednja pravila:

- prva vrstica vsebuje formalno ime<sup>5</sup> in heksadecimalni naslov registra (v RAM),
- druga vrstica vsebuje zaporedne številke posameznih bitov,
- tretja vrstica vsebuje formalna imena posameznih bitov ali skupin bitov,
- četrta vrstica vsebuje stanje bita po resetiranju MC68332.

CSORx

| 15    | 14   | 13  | 12 | 11   | 10 | 9            | 8 | 7 | 6 | 5     | 4 | 3   | 2 | 1    | 0 |
|-------|------|-----|----|------|----|--------------|---|---|---|-------|---|-----|---|------|---|
| MODE  | BYTE | R/W |    | STRB |    | <b>DSACK</b> |   |   |   | SPACE |   | IPL |   | AVEC |   |
| RES:0 |      |     |    |      |    |              |   |   |   |       |   |     |   |      |   |
| 0     | 0    | 0   | 0  | 0    | 0  | 0            | 0 | 0 | 0 | 0     | 0 | 0   | 0 | 0    | 0 |

CSBARx

| 15   | 14  | 13  | 12  | 11  | 10  | 9   | 8   | 7   | 6   | 5   | 4   | 3   | 2 | 1     | 0 |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|-------|---|
| A23  | A22 | A21 | A20 | A19 | A18 | A17 | A16 | A15 | A14 | A13 | A12 | A11 |   | BLKSZ |   |
| RES: |     |     |     |     |     |     |     |     |     |     |     |     |   |       |   |
| 0    | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0   | 0 | 0     | 0 |

*Slika 5. 7: Registra CSORx in CSBARx*

Pomen nekaterih polj v CSORx registrih:

#### BYTE:

- 00 - CS razveljavljen
- 01 - spodnji byte
- 10 - zgornji byte
- 11 - oba byta

#### R/W:

- 00 - rezervirano
- 01 - beri (read)
- 10 - piši (write)
- 11 - oboje

#### DSACK:

0000 - 0 čakalnih stanj (wait states)

.....

1101 - 13 čakalnih stanj

**BYTE** področje uporabljamo pri naslavljjanju 16-bitne periferije. V tem primeru moramo določiti, ali naslavlja CS spodnji ali zgornji byte besede ali pa oba (glej primer na sliki 5. 9).

<sup>5</sup> Mnemonik registra (v tem primeru CSORx) in mnemoniki posameznih bitov ali skupin bitov (tukaj MODE, BYTE itd.) nimajo nobenega konkretnega pomena. Njihov namen je le opozarjanje na funkcijo, ki jim je namenjena. V programu moramo tem simbolom dodeliti dejanske naslove (npr. register CSOR3 naslavljamo s tem simbolom le, če smo mu dodelili naslov Yffa5a<sub>HEX</sub>, MODE bit je le 15. bit tega registra itn. Glej primer programiranja v pogl. 7.2).

Pri naslavljjanju je LSB naslovnega vodila (ADR0) zadolžen za ločitev med spodnjim in zgornjim oz. sodim in lihim bytom.

Naslavljanje enot, ki dovoljujejo le branje (npr. ROM pomnilnikov, A/D pretvornikov itd.), določimo v polju **R/W** z bitoma 01. Kombinacija 10 aktivira CS le pri vpisu v enoto (npr. D/A pretvornik), če pa se bo signal za izbiro čipa aktiviral pri obeh operacijah (npr. ob naslavljjanju RAM pomnilnikov), postavimo vsebino polja na 11.

Prilagajanje delovanja MC68332 počasnejši periferiji lahko dosežemo z uvajanjem čakalnih stanj. V polju **DSACK** definiramo njihovo trajanje z mnogokratnikom periode sistemske ure.

**CSBARx** register določa naslov in dolžino pomnilniškega bloka, ki ga želimo naslavljati (slika 5. 8).

| Polje BLKSIZE | Dolžina bloka | Upoštevani biti naslovnega vodila |
|---------------|---------------|-----------------------------------|
| 000           | 2 K           | A23 - A11                         |
| 001           | 8 K           | A23 - A13                         |
| 010           | 16 K          | A23 - A15                         |
| 011           | 64 K          | A23 - A16                         |
| 100           | 128 K         | A23 - A17                         |
| 101           | 256 K         | A23 - A18                         |
| 110           | 512 K         | A23 - A19                         |
| 111           | 1 M           | A23 - A20                         |

*Slika 5. 8: Kode za register CSBARx*

Biti 15 - 3 določajo naslov (biti na naslovnem vodilu A23 - A11), na katerem se nahaja spominski blok dolžine, ki je določena s poljem BLKS (angl. Block Size). Če želimo na naslov F00000<sub>HEX</sub> postaviti 8 Kbytni pomnilnik s slike 5. 3, moramo BLKS maskirati v stanje 001. Naslov bloka pomnilnika vpišemo v naslovno polje. Iz tabele je razvidno, da bodo pri definiranju tega naslova sodelovali le biti A23 - A13, kar smo ugotovili tudi v primeru iz podpoglavlja 5.5.1. Pri tem bita A11 in A12 nimata nobenega vpliva, saj ju, skupaj s biti A0 - A10 rabimo za določitev interne lokacije v RAM.

Kombinacija bitov za omenjeni primer je:

$$(\text{CSBAR}) = 1111\ 0000\ 000x\ x001_{\text{BIN}}.$$

Pri naslavljjanju katerekoli bytne lokacije med F00000<sub>HEX</sub> - F01FFF<sub>HEX</sub> se bo CS, ki smo ga določili za ta RAM, avtomatično postavil v aktivno stanje (0). Na ta način smo se izognili potrebi po hardverskem določanju CS.

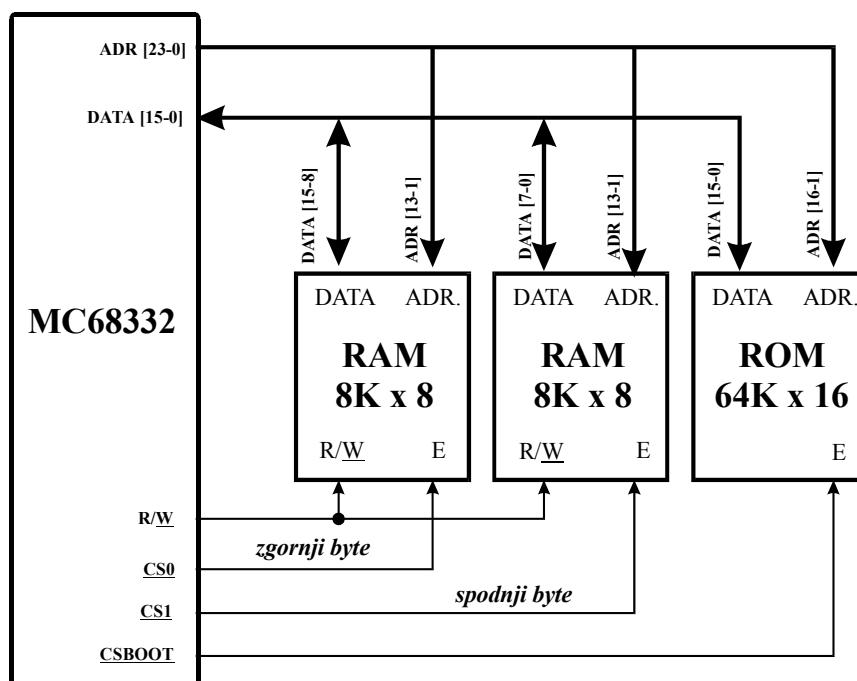
#### 5.5.2.1 Primer naslavljanja pomnilnikov

Tipičen primer naslavljanja pomnilnikov ob uporabi CS izhoda ter vodil kaže slika 5. 9 [24]. V tem primeru imamo opravka z dvema RAM pomnilnikoma velikosti 8 Kbyte ( $8\text{K} \times 8\text{ bit}$ ) in enim "boot" ROM (ali EPROM) pomnilnikom velikosti 64 Kword ( $64\text{K} \times 16\text{ bit}$ ).

Podatkovno vodilo ROM je povezano z ustreznimi biti zunanjega 16-bitnega podatkovnega vodila MC68332. Podatkovno vodilo prvega RAM je povezano z MSByte podatkovnega vodila mikrokrmlnika, katerega spodnja polovica (LSByte) pa je povezana z drugim RAM pomnilnikom. Naslovni biti MC68332 (A13-A1) so povezani z A12 - A0 pomnilnikov, torej so premaknjeni za en bit v levo!

Razlog za takšne povezave je enostaven. Mikrokrmlnik bo iz perifernih pomnilnikov bral podatke dolžine ene besede (16 bitov). Pri vsaki besedi, ki jo preberamo iz RAMa, se bo ena polovica nahajala v enem, druga pa v drugem čipu. Torej smo s tako konfiguracijo dosegli ekvivalent RAM pomnilnika velikosti 8 Kword. Z ozirom na hkratno naslavljanje obih RAM pomnilnikov<sup>6</sup> moramo tudi CS0 in CS1 prožiti hkrati. Zaradi naslavljanja bytov z različno težo moramo BYTE področje CSOR1 registra postaviti v stanje 01<sub>BIN</sub> (CS1 naslavlja spodnji byte), CSOR0 pa v 10<sub>BIN</sub> (CS0 naslavlja zgornji byte)<sup>7</sup>.

Naj bo vsebina CSBAR0 in CSBAR1 enaka 1001<sub>HEX</sub>. S tem smo za RAM rezervirali področje  $2 \times 8\text{ Kbytov}$ , ki se začne na naslovu 100000<sub>HEX</sub>.



*Slika 5. 9: Primer naslavljanja zunanjih pomnilnikov*

ROM pomnilnik vsebuje celice dolžine ene besede, kar pomeni, da posameznih bytov ne moremo nasloviti. Zato so tudi tukaj biti naslovnega mikrokrmlnika ADR16-1 povezani z biti

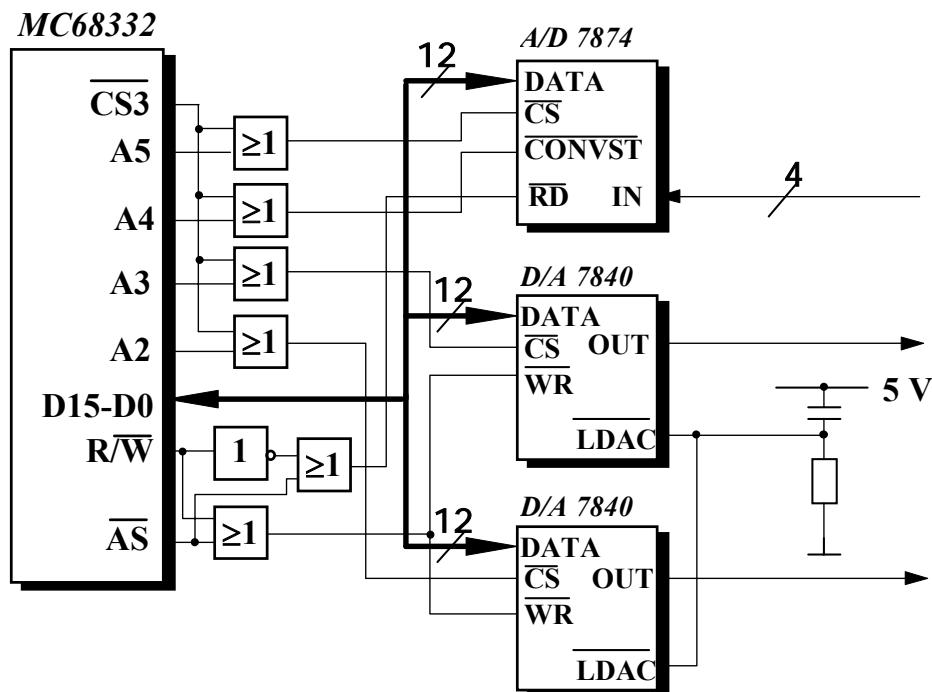
<sup>6</sup> PAZI! V tem primeru hkratno naslavljanje ne pomeni tudi konflikta na podatkovnem vodilu, saj pošilja vsak čip hkrati podatke po svoji polovici vodila.

<sup>7</sup> CS signali mikrokontrolerja so povezani z vhodi E (angl. enable - omogočiti) na čipih (glej tudi pogl. 12.1).

ADR15-0 pomnilnika. (CSBARBT) = 0003<sub>HEX</sub>, kar določa aktiviranje CSBOOT za naslavljjanje področja velikosti 128 Kbyte = 64 Kword na začetnem naslovu 0.

### 5.5.2.2 Primer naslavljjanja majhnih enot

Najmanjše področje, ki ga na ta način naslovimo, je 2 K (slika 5. 8). Če pa hočemo nasloviti manjše področje (npr. A/D ali D/A pretvornik), moramo preostale bite (A10 - A0) ustrezno dekodirati. Slika 5. 10 kaže možno rešitev izbire dvobytnih lokacij brez potrebe po dekodirjanju.



*Slika 5. 10: Uporaba CS3 pri naslavljjanju dvobytnih lokacij*

V tem primeru naslavljamo tri čipe: 12-bitni A/D pretvornik in dva 12-bitna D/A pretvornika (pogl. 12.4). Pri izbiri čipov bo sodeloval CS3. V fazi inicializacijskega dela programa bomo registra za konfiguriranje CS postavili v naslednje stanje:

$$(\text{CSBAR3}) = 0500_{\text{HEX}} = 0000\ 0101\ 0000\ 0000_{\text{BIN}}$$

$$(\text{CSOR3}) = 7b30_{\text{HEX}} = 0111\ 1011\ 0011\ 0000_{\text{BIN}}.$$

S prvim registrom smo določili osnovni naslov 050000<sub>HEX</sub> najmanjšega spominskega bloka (2 K). Drugi register pa definira način naslavljanja: asinhronski prenos, možnost posamičnega naslavljanja sodega in lihega byta, proženje CS pri branju (read) in vpisovanju (write) ob veljavnem naslovu na vodilu (AS), 12 čakalnih stanj (wait states) itd. Znotraj bloka 2 K sodelujejo pri naslavljjanju naslovni biti A10-A0 (slika 5. 11).

| 2. nibble |    |    | 1. nibble |    |    |    | 0. nibble |    |       |       | Izbrani čip         |
|-----------|----|----|-----------|----|----|----|-----------|----|-------|-------|---------------------|
| A10       | A9 | A8 | A7        | A6 | A5 | A4 | A3        | A2 | A1    | A0    |                     |
| X         | X  | X  | X         | X  | 0  | 1  | 1         | 1  | $X^8$ | $Y^9$ | <u>CS</u> AD7874    |
| X         | X  | X  | X         | X  | 1  | 0  | 1         | 1  | X     | Y     | <u>CONVST</u>       |
| X         | X  | X  | X         | X  | 1  | 1  | 0         | 1  | X     | Y     | <u>CS</u> 1. DA7840 |
| X         | X  | X  | X         | X  | 1  | 1  | 1         | 0  | X     | Y     | <u>CS</u> 2. DA7840 |

**Slika 5. 11: Kombinacije bitov A10-A0 za izbiro posameznih čipov s slike 5. 10**

Izbrana arhitektura s slike 5. 10 namesto zapletenega dekodiranja enajstih bitov uporablja le štiri bite A5-A2. Posamezni čipi bodo naslovljeni le, če je **eden** izmed teh bitov 0, **vsi ostali** pa 1. Biti A10-A6 in A1 lahko imajo **poljubna stanja**. Izjema je le bit A0, ki je 0, če hočemo nasloviti spodnjih osem bitov pretvornika oz. 1, če naslavljamo zgornje štiri bite. To dejansko pomeni, da npr. A/D pretvornik naslovimo na več načinov (05001e<sub>HEX</sub>, 0500dc<sub>HEX</sub>, 05001d<sub>HEX</sub>, 05005d<sub>HEX</sub> itd.). Pomembna je le kombinacija 0111 na nožicah A5-A2!

Čeprav je zgornji način nekoliko nekonvencionalen, izpolnjuje osnovno zahtevo, ki pravi, da z enim naslovom ne smemo nasloviti več čipov. V takšnih arhitekturah lahko zgornjo idejo uporabimo ob dodatnem sodelovanju dekoderjev. Osnovna pomanjkljivost opisanega pristopa je v tem, da smo področje 2 Kbytov spominske mape MC68332 uporabili za naslavljjanje le  $2 \times 4 = 8$  bytov.

<sup>8</sup> X je poljubna vrednost: 1 ali 0.

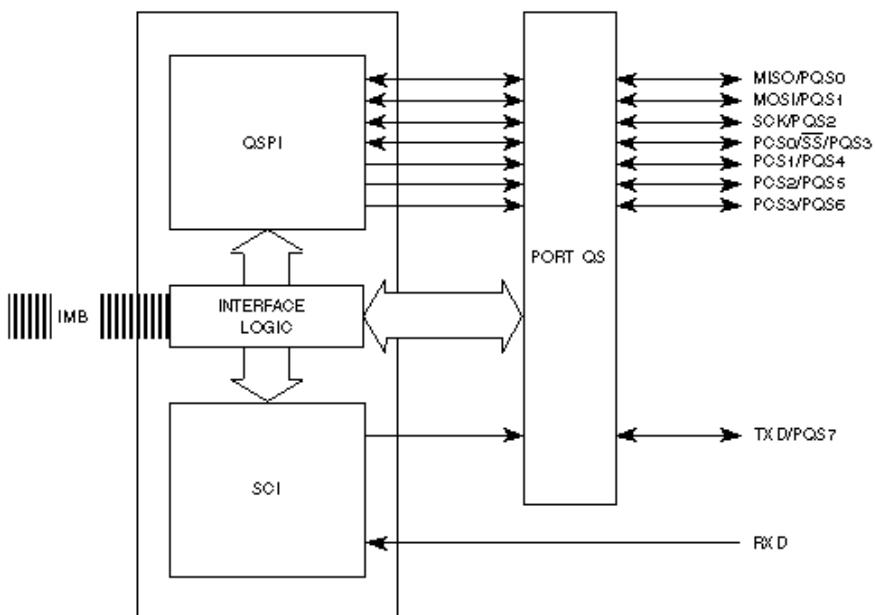
<sup>9</sup> Y je 0, če naslavljamo sodi bajt (spodnjih osem bitov pretvornika) in 1, če naslavljamo lihi bajt (zajeti so štirje zgornji biti pretvornika).

## 6. MODUL ZA SERIJSKO KOMUNIKACIJO - QSM

**QSM** (Queued Serial Module) je modul, ki vsebuje dva vmesnika za serijsko komunikacijo mikrokrmlilnika MC68332: **QSPI** (Queued Serial Peripheral Interface) in **SCI** (Serial Communication Interface).

QSPI je *full-duplex*<sup>1</sup> sinhronski serijski vmesnik za komunikacijo s periferijo in ostalimi mikroprocesorji. SCI je full-duplex univerzalni asinhronski zaporedni vmesnik (angl. Universal Asynchronous Receiver/Transmitter - UART). Slika 6. 1 kaže blokovno shemo QSM.

Značilnost *sinhronske komunikacije* je sinnhronizacija prenosa bitov s posebnim časovnim signalom. Pri *asinhronki komunikaciji* se prenos podatkov začne v poljubnem trenutku, ob spremembi t.i. *start bita*.



Slika 6. 1: Blokovna shema QSM

Delovanje QSM krmili CPU prek globalnih registrov, QSPI in SCI nadzornih in statusnih registrov ter QSPI RAM, v katerega vpisujemo ali beremo vhodno/izhodne podatke (slika 6. 2).

Prek QSM konfiguracijskega registra (QMCR) določamo osnovne nastavitev QSM (omogočanje delovanja, prioriteto QSM prekinitev z ozirom na ostale module MC68332 itd.), QILR pa določa interni prekinitveni prioriteti QSPI in SCI.

<sup>1</sup> Izraz izvira iz brezžične komunikacijske tehnike in označuje način komuniciranja med dvema uporabnikoma: *simplex* kanal omogoča enosmerno komunikacijo, *half-duplex* izmenično dvosmerno, *full-duplex* pa hkratno dvosmerno komunikacijo.

|             | 15     | 8 7         | 0    |
|-------------|--------|-------------|------|
| Yffc00      |        | QMCR        |      |
| Yffc02      |        | QTEST       |      |
| Yffc04      | QILR   |             | QIVR |
| Yffc06      |        | REZERVIRANO |      |
| Yffc08      |        | SCCR0       |      |
| Yffc0a      |        | SCCR1       |      |
| Yffc0c      |        | SCSR        |      |
| Yffc0e      |        | SCDR        |      |
| Yffc10      |        | REZERVIRANO |      |
| Yffc12      |        | REZERVIRANO |      |
| Yffc14      | REZER. |             | QPDR |
| Yffc16      | QPAR   |             | QDDR |
| Yffc18      |        | SPCR0       |      |
| Yffc1a      |        | SPCR1       |      |
| Yffc1c      |        | SPCR2       |      |
| Yffc1e      | SPCR3  |             | SPSR |
| Yffc20 - ff |        | REZERVIRANO |      |
| Yffd00 - 1f |        | REC. RAM    |      |
| Yffd20 - 3f |        | TRAN. RAM   |      |
| Yffd40 - 4f |        | COMD. RAM   |      |

**Slika 6. 2: Spominska mapa QSM**

Razen inicializacije QSM je naloga CPU pri zaporednem prenosu le branje prejetih ali vpisovanje podatkov za pošiljanje. Samo komunikacijo opravlja QSM neodvisno od CPU in ga na tačin bistveno razbremeni.

## 6.1 Registri za krmiljenje QSM nožic

Nožice QSM (razen SCK) lahko uporabljamo kot navadne binarne vhode/izhode (s čimer povečamo skupno število vhodov in izhodov), ali pa jim dodelimo njihovo primarno funkcijo, to je serijsko komunikacijo<sup>2</sup>. Za izbiro med dvema režimoma uporabljamo registre QPDR (QSM Port Data Register), QPAR (QSM Pin Assignment Register) in QDDR (QSM Data Direction Register) (slika 6. 3).

QPDR določa funkcijo nožic: resetiranje bitov (stanje 0) določa ustrezne nožice kot splošne binarne vhode/izhode (I/O), setiranje (signal 1) pa jim dodeljuje njihovo prvotno funkcijo QSM nožic. Nožicam, ki delujejo kot vhodi/izhodi določamo smer pretoka s pomočjo registra QDDR; setiranje jih definira kot izhode, resetiranje kot vhode. Za postavljanje tako določenega vhoda v želeno stanje (1 ali 0) ali čitanje stanja vhodne nožice skrbi register QPDR.

Če so nožice v funkciji serijskega prenosa (QSM ali QSPI) obdržijo svojo prvotno funkcijo.

<sup>2</sup> Nožice, ki v določenem trenutku niso v funkciji serijskega prenosa (tudi, če ostale pri tem sodelujejo), so lahko definirane kot vhodi ali izhodi. Le SCK je pri sinhronskem prenosu nujno potreben.

QPDR

Naslov: Yffc15<sub>HEX</sub>

| 15          | 8           | 7            | 6            | 5            | 4                           | 3           | 2            | 1            | 0 |
|-------------|-------------|--------------|--------------|--------------|-----------------------------|-------------|--------------|--------------|---|
| REZERVIRANO | D7<br>(TXD) | D6<br>(PCS3) | D5<br>(PCS2) | D4<br>(PCS1) | D3<br>(PCS0/<br><u>SS</u> ) | D2<br>(SCK) | D1<br>(MOSI) | D0<br>(MISO) |   |

RES:

0 0 0 0 0 0 0 0 0

QPAR

Naslov: Yffc16<sub>HEX</sub>

| 15 | 14   | 13   | 12   | 11                 | 10 | 9    | 8    | 7                      | 0 |
|----|------|------|------|--------------------|----|------|------|------------------------|---|
| 0  | PCS3 | PCS2 | PCS1 | PCS0/<br><u>SS</u> | 0  | MOSI | MISO | xxxxxxxxxxxxxxxxxxxxxx |   |

RES:

0 0 0 0 0 0 0 0

0 = splošno uporabni vhodi/izhodi

1 = QSPI modul

QDDR

Naslov: Yffc17<sub>HEX</sub>

| 15                   | 8   | 7    | 6    | 5    | 4                  | 3   | 2    | 1    | 0 |
|----------------------|-----|------|------|------|--------------------|-----|------|------|---|
| xxxxxxxxxxxxxxxxxxxx | TXD | PCS3 | PCS2 | PCS1 | PCS0/<br><u>SS</u> | SCK | MOSI | MISO |   |

RES:

0 0 0 0 0 0 0 0 0

0 = vhod

1 = izhod

*Slika 6. 3: Registri za določanje funkcije QSM*

## 6.2 Sinhronika serijska komunikacija (QSPI)

Mikrokrmilniški modul QSPI je namenjen sinhronski serijski komunikaciji z ustreznou periferno enoto (npr. serijskim A/D pretvornikom) ali drugim mikroračunalnikom. QSPI je izboljšana inačica Motorolinega standarda SPI (Serial Peripheral Interface), ki je prisoten tudi na manj zmogljivih mikrokrmilnikih (npr. na 8-bitnem MC68HC11). Osnovne značilnosti SPI so:

- Sinhronski trižilni Full-Duplex ali štirižilni Half-Duplex prenos.
- Master ali Slave način obratovanja.
- Programirljiva hitrost prenosa pri Master obratovanju.
- Možnost generiranja prekinitve po končanem prenosu.
- Preprosta povezava z razširitvenimi enotami: A/D pretvorniki, EEPROMi, prikazovalniki (display) itd.

QSPI vsebuje nekatere dodatne funkcije:

- Programirljivo vrsto (angl. queue<sup>3</sup>) - do 16 vnaprej programiranih prenosov.
- Programirljivo generiranje signalov za izbiro čipov (CS) - štiri nožice lahko naslovijo do 16 perifernih vezij.
- Preskočni (angl. wraparound) način prenosa - možnost avtomatičnega cikličnega naslavljjanja periferije (npr. A/D pretvornika) brez posredovanja CPU.
- Programirljiva dolžina serijskega podatka (od 6 do 16 bitov) itd.

Pri serijski povezavi QSPI modula s periferno enoto ali drugim QSPI modulom se moramo odločiti, kdo bo imel "glavno besedo" pri komunikaciji (angl. *Master* - nadrejeni, dobesedno: gospodar). Vse ostale enote (ena ali več) se morajo prilagoditi njegovem "diktatu" in imajo podrejeno vlogo sužnjev (angl. *Slave*)<sup>4</sup>. Izbiro med Master ali Slave načinom delovanja opravimo z MSTR bitom v registru SPCR0 (1 - QSPI je sistemski Master, 0 - QSPI je Slave). Prenos podatkov krmilimo preko naslednjih nožic:

| Ime nožice  | Mnemonik<br>(simbol) | Način delovanja   | Funkcija   |
|---|----------------------|-------------------|--|
| Master In Slave Out   | MISO                 | Master:<br>Slave: | Serijski vhodni podatki v QSPI<br>Serijski izhodni podatki iz QSPI |
| Master Out Slave In   | MOSI                 | Master:<br>Slave: | Serijski izhodni podatki iz QSPI<br>Serijski vhodni podatki v QSPI |
| Serijska ura (clock)  | SCK <sup>1</sup>     | Master:<br>Slave: | Izhodni takt iz QSPI<br>Vhod v QSPI                                |
| Periferni <u>CS</u>   | PCS3-PCS1            | Master:           | Izhod; izbiro periferije   |
| Periferni <u>CS</u> <sup>2</sup><br>ali izbiro Slave <sup>3</sup> | PCS0/<br><u>SS</u>   | Master:<br>Slave: | Izhod; izbiro periferije<br>Vhod; izbere QSPI                      |
| Izbira Slave <sup>4</sup>   | <u>SS</u>            | Master:           | Lahko povzroči konflikt  |

#### Pripombe:

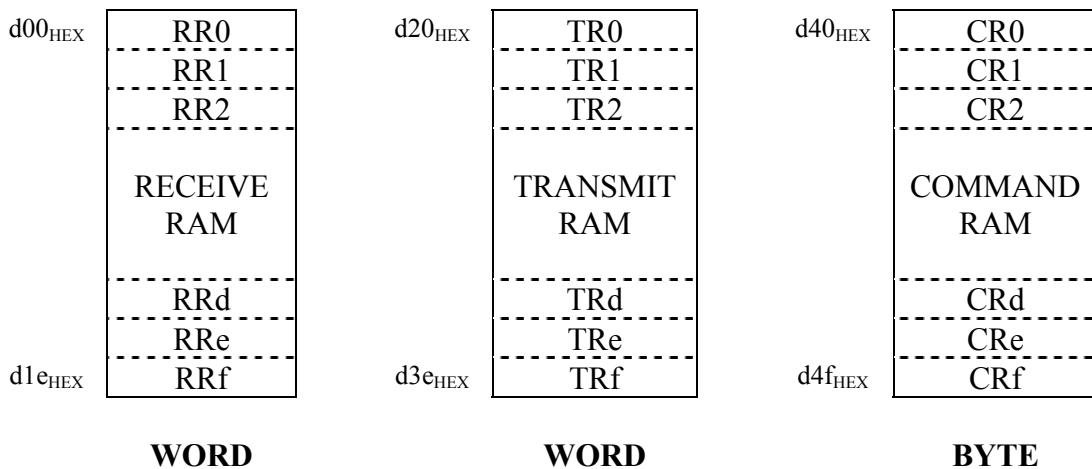
1. Vse QSPI nožice (razen SCK) lahko uporabljamo kot vhode/izhode, če jih QSPI ne uporablja za prenos.
2. Izhod, če QSPI deluje kot Master.
3. Vhod, če QSPI deluje kot Slave.
4. Vhod, če QSPI deluje kot Master; koristen, če imamo sistem s več Master enotami.

Pri snovanju Master-Slave konfiguracije moramo skrbeti za ustrezno povezavo med enotami ter definiranje QSPI nožic na obeh straneh komunikacijske verige. Tako bo na primer Master preko MOSI nožice pošiljal izhodne serijske podatke prek QSPI, pri Slave enoti pa bodo na tej nožici prihajali vhodni podatki. Primer povezave s podrejenim D/A pretvornikom je prikazan v pogl. 6.2.1.

Vhodni podatki (pri podrejenem) ali izhodni podatki (pri nadrejenem) se vpisujejo v QSPI RAM. MC68332 ima možnost vpisovanja ali zajemanja šestnajstih 16-bitnih besed (slika 6. 4). V to področje lahko posega tudi CPU in bere ali vpisuje podatke za prenos. Konkretno lokacije posameznih RAM področij si lahko ogledamo tudi na sliki 6. 2.

<sup>3</sup> Vrsta je podatkovna struktura, kjer se vhodni ali izhodni komunikacijski podatki nalagajo v posebno pomnilniško področje, ki deluje po FIFO principu (angl. First In First Out: prvi noter - prvi ven).

<sup>4</sup> Običajno imamo le enega nadrejenega in enega ali več podrejenih (npr. več pretvornikov ali procesorjev), obstaja pa tudi možnost definiranja t.i. multimaster sistema z več nadrejenimi enotami.



**Slika 6. 4: Organizacija QSPI RAM**

Vhodni podatki se shranjujejo v sprejemni RAM (angl. receive RAM), podatke, ki jih Master pošilja proti periferiji, pa vpisuje CPU v oddajni RAM (angl. transmit RAM).

Prenos vsake izmed posameznih šestnajstih besed v ali iz katerega komunikacijskega partnerja krmilimo preko krmilnega ali ukaznega RAM (angl. command control RAM). Smer in način pošiljanja določamo prek setiranja ustreznega bita v ukaznem RAM (slika 6. 5).

|      |       |    |      |      |      |      |       | Naslov: Yffd40 <sub>HEX</sub> |
|------|-------|----|------|------|------|------|-------|-------------------------------|
| 7    | 6     | 5  | 4    | 3    | 2    | 1    | 0     |                               |
| CONT | BITSE | DT | DSCK | PCS3 | PCS2 | PCS1 | PCS0* |                               |
| -    | -     | -  | -    | -    | -    | -    | -     |                               |
| -    | -     | -  | -    | -    | -    | -    | -     |                               |
| CONT | BITSE | DT | DSCK | PCS3 | PCS2 | PCS1 | PCS0* |                               |

Naslov: Yffd4f<sub>HEX</sub>

|                 |                        |
|-----------------|------------------------|
| COMMAND CONTROL | PERIPHERAL CHIP SELECT |
|-----------------|------------------------|

**Slika 6. 5: Pomen bitov v krmilnem RAM (\* PCS0 ima dvojno funkcijo: PCS0/SS)**

Pomen bitov:

PCS3-PCS0/SS - setiranje enega izmed teh štirih bitov ali njihove kombinacije. Nožice so povezane s CS vhodi perifernih enot, s katerimi komuniciramo (z ustreznim dekodiranjem lahko naslovimo do  $2^4 = 16$  enot). S tem v Master režimu določamo konkretno enoto (ali pa več enot hkrati), ki ji pošiljamo podatke.

CONT - nadzira spremembo stanja aktualnega PCSx bita po končanem prenosu tekoče besede (1 - ostaja enak, 0 - prevzame stanje, ki je definirano v registru QPDR).

BITSE - število bitov, ki jih pošiljamo ali sprejmemo (1 - število bitov, določeno v registru SPCR0, 0 - 8 bitov)<sup>5</sup>.

DT - zakasnitev med dvema signaloma za izbiro perifernega vezja PCS in SCK (1 - zakasnitev določena v DSCKL polju registra SPCR1, 0 - zakasnitev enaka 1/2 SCK).

Poleg omenjenih registrov je treba pred začetkom prenosa definirati tudi druge parametre, ki odločilno vplivajo na njegov potek. Tako je npr. serijski prenos (preko MOSI ali MISO nožice) sinhroniziran s frekvenco časovnega signala na nožici SCK. Izbira ustrezne kombinacije bitov v SPBR polju registra SPCR0 omogoča frekvenco SCK od 33 kHz do 4,19 MHz (pri sistemski frekvenci MC68332 16,77 MHz). Pri izbiri frekvence prenosa izhajamo iz karakteristik perifernih enot, s katerimi komuniciramo. Tako npr. večina A/D in D/A pretvornikov lahko pošilja zaporedne podatke s frekvenco do nekaj sto kHz. Hitrost prenosa je odvisna tudi od okolja (problem motenj pri visokih frekvencah).

### 6.2.1 Serijski A/D in D/A pretvorniki (primer sinhronske serijske komunikacije)

“Klasični” paralelni A/D in D/A pretvorniki komunicirajo s procesorjem preko paralelnega vmesnika (pogl. 12.4). Po naslavljjanju se vsi podatkovni biti (npr. 16 bitov) pošljejo hkrati - v enem samem časovnem ciklu - na podatkovno vodilo. Na ta način dosežemo maksimalno možno hitrost prenosa. Po drugi strani narekuje takšen pristop v določenih aplikacijah izdelavo dokaj zapletenega vezja. Mikroprocesor in pretvornik si morata deliti podatkovno vodilo (npr. 16-bitno) in ostale signale (CS, AS ...). V primerih, ko imamo na voljo le enoplastno ploščico, na kateri se nahaja veliko integriranih vezij, pride kmalu do velike “gneče”, saj se številne povezave med seboj križajo.

Po drugi strani je poglavitna lastnost serijske komunikacije prenos po eni ali dveh linijah, kar bistveno zmanjša število povezav. Osnovna pomanjkljivost takšne izvedbe je nižja hitrost prenosa, saj se biti pošiljajo zapovrstjo. Čas, potreben za prenos šestnajstih bitov, bo daljši od 16 urinih ciklov. V povezavah preko asinhronskega vmesnika EIA-232 (prej RS-232C) je tak prenos prepočasen<sup>6</sup> in v aplikacijah z A/D in D/A pretvorniki pride le redkokdaj v poštev. Serijska komunikacija s pretvorniki je smiselna le, če so hitrosti prenosa nekaj sto Kbps (angl. bits per second - bps<sup>7</sup>). To zahteva posebne pretvornike ter poseben protokol in vmesnik tipa SPI oz. QSPI pri Motorolinih procesorjih, Microwire itd.

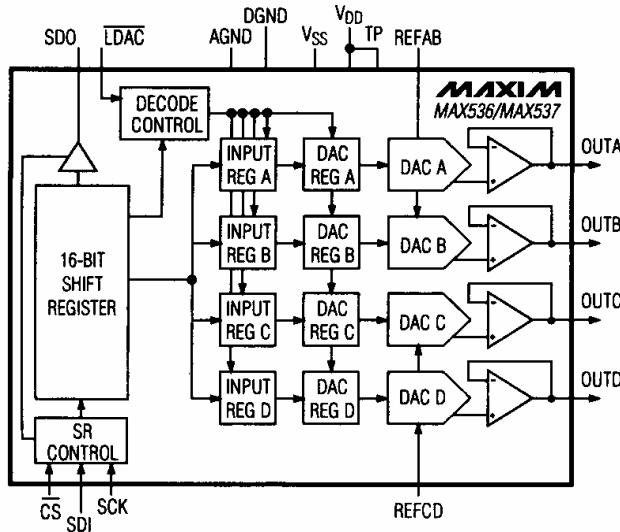
---

<sup>5</sup> Odvisno od zahtev sistema. Pri komunikaciji z D/A pretvornikom MAX 536 npr., pošiljamo 16-bitni podatek (glej tudi pogl. 6.2.1)

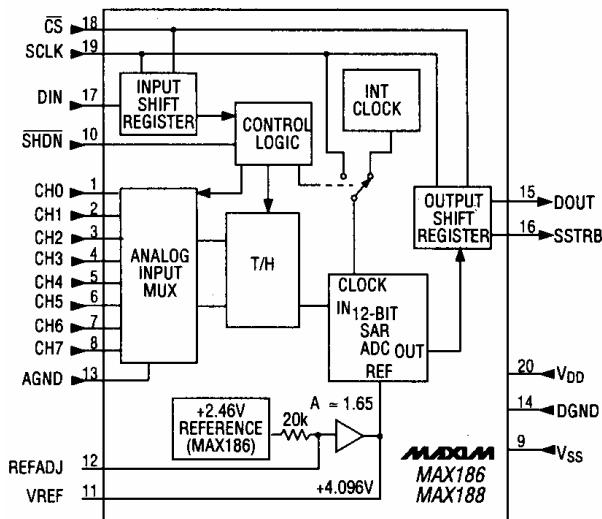
<sup>6</sup> Npr. pri 19200 bps pošljemo 19200 bitov v sekundi. Glede na to, da se v paketu poleg šestnajstih podatkovnih bitov (moramo jih razbiti na vsaj dva paketa po 8 bitov) nahajajo še startni bit, eden ali dva stop bita in paritetni bit, bo celoten prenos ene besede trajal več kot 1 ms. Ob upoštevanju dejstva, da traja paralelna komunikacija s pretvornikom nekaj  $\mu$ s, hitro ugotovimo nesmotrnost uporabe takega pristopa.

<sup>7</sup> Za označevanje hitrosti prenosa se uporablja tudi izraz *baud* (kratica Bd), ki izvira iz telegrafije. Zaradi možnih različnih tolmačenj tega izraza se raje zamenjuje z izrazom bits per second ali bytes per second.

V tem poglavju si bomo ogledali tipična predstavnika serijskih pretvornikov firme MAXIM: 4-kanalni, 12-bitni D/A pretvornik (MAX536, slika 6. 6) in 8-kanalni 12-bitni A/D pretvornik (MAX186, slika 6. 7), v povezavi s QSPI vmesnikom.



*Slika 6. 6: Blokovna shema serijskega D/A pretvornika MAX 536/537*



*Slika 6. 7: Blokovna shema serijskega A/D pretvornika MAX 185/186*

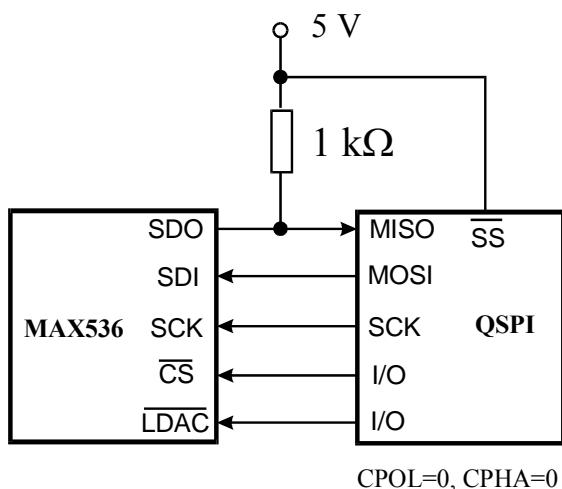
Osnovno povezavo D/A pretvornika s QSPI modulom mikrokrumilnika (npr. MC68HC11 ali MC68332) kaže slika 6. 8. Pretvornik lahko sprejeme in celo pošilja serijske podatke. Zato je treba nožico za vhodne serijske podatke (izhoda iz pretvornikov: SDO - Serial Data Output ali DOUT za A/D na sliki 6. 7) povezati z vhodno nogico QSPI (MISO - Master Input Slave Output). Za podatke, ki tečejo v nasprotni smeri, povežemo nožici SDI (Serial Data Input ali DIN za A/D pretvornik) in MOSI (Master Output Slave Input). Vezava izhoda SDO na QSPI ni nujno

potrebna (lahko ga uporabimo za verifikacijo podatkov), na SDI nogico pa pripeljemo digitalno vrednost, ki jo želimo pretvoriti v analogno.

Vhod CS ima enako funkcijo, kot pri vseh ostalih integriranih vezjih: le pri aktivnem CS (logični signal 0) je omogočena komunikacija s pretvornikom. Neaktivni CS zapahne (angl. latch) podatke v vmesnih registrih.

Serijski podatki “potujejo” v sinhronizmu s frekvenco, ki jo narekuje “serijska ura” SCK (angl. Serial Clock, SCLK za A/D pretvornik na sliki 6. 7). V konkretnem primeru je maksimalna možna frekvenca na SCK 10 MHz.

Aktivni signal na vhodu LDAC omogoča prenos podatkov iz vhodnih registrov (angl. Input Registers) posameznih kanalov v DAC registre (glej sliko 6. 6). Povezava tega signala s QSPI, podobno kot tudi SDO, ni nujno potrebna. V tem primeru ga vežemo na maso in tako omogočimo takojšnji prehod signalov iz vhodnih v DAC registre.

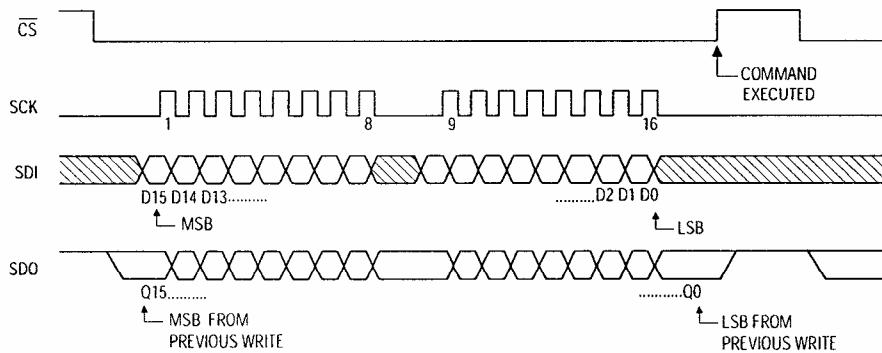


**Slika 6. 8: Povezava D/A pretvornika s serijskim vmesnikom MAX536 in QSPI modula mikrokrnilnika**

Iz povedanega je razvidno, da je serijsko povezavo med pretvornikom in procesorjem možno izvesti že s tremi vodniki (brez SDO in LDAC), kar je bistveno manj kot pri paralelnih A/D in D/A pretvornikih!

#### 6.2.1.1 Časovni potek in format podatkov

Časovni potek minimalne (3-žične) serijske povezave kaže slika 6. 9. QSPI mora najprej poskrbeti za izbiro pretvornika (aktivni signal CS). Po določenem, vnaprej predpisanim času se začne serijski prenos šestnajstih podatkov v sinhronizmu z uro (SCK).



**Slika 6. 9: Časovni diagram serijskega prenosa preko treh vodnikov**

Serijski podatek 12-bitnega MAX536 je sestavljen iz dveh naslovnih bitov, dveh nadzornih bitov in dvanajstih podatkovnih bitov (slika 6. 10). Različne kombinacije prvih dveh bitov ( $2^2 = 4$ ) določajo enega izmed štirih D/A pretvornikov. Naslednja dva bita omogočata različne časovne sekvence pretokov med vhodnimi registri in DAC registri, dejanska digitalna vrednost, ki jo želimo pretvoriti v izhodno napetost (od 0 do  $V_{REF}$  pri izbiri unipolarnega izhoda in od  $-V_{REF}$  do  $+V_{REF}$  pri bipolarnem izhodu), pa je zapisana v podatkovnih bitih.

| 16 bitov serijskega podatka     |               |                          |    |
|---------------------------------|---------------|--------------------------|----|
| Naslovna bita                   | Nadzorna bita | Podatkovni biti          |    |
| A1                              | A0            | C1                       | C0 |
| ← 4 naslovni in nadzorni biti → |               | ← 12 podatkovnih bitov → |    |

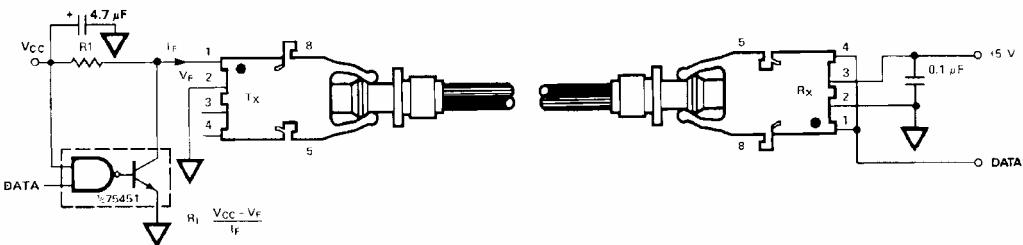
**Slika 6. 10: Format serijskega podatka MAX536**

#### 6.2.1.2 Prenos podatkov prek optičnih kablov

Hiter prenos podatkov po električnih vodnikih je povezan s številnimi težavami, še posebej če so povezave dolge in potekajo skozi okolje, ki generira različne motnje. Vse to vpliva na zanesljivost in omejuje največjo hitrost prenosa.

Svetloba je neobčutljiva na omenjene vplive in je zato njen prenos preko fleksibilnih optičnih vlaken ali kablov (steklenih ali plastičnih) optimalna rešitev za hitro in zanesljivo pošiljanje digitalnih podatkov.

V praksi obstaja celo paleta optičnih kablov, ki zagotavljajo ekstremno hiter prenos na velike razdalje. S stališča večine projektantov mikroracunalniških sistemov za krmiljenje industrijskih procesov, kjer sta sprejemnik in oddajnik oddaljena nekaj metrov in je potrebna le zmerna hitrost prenosa (nekaj Mbps), ponuja trg cenejše izvedbe, kot je npr. Versatile Link firme Hewlett Packard (slika 6. 11).



**Slika 6. 11: HP Versatile Link HFBR-15X1/25X1**

Povezava (angl. link) je sestavljena iz optičnega kabla s končnikoma ter oddajnega (Tx) in sprejemnega (Rx) plastičnega modula, ki sta hkrati konektorja. Maksimalna hitrost prenosa je 5 Mbps, kar presega možnosti večine pretvornikov, maksimalna dolžina kabla pa je nekaj deset metrov. Velikost potrebnega vhodnega toka  $I_F$  se z razdaljo eksponencialno povečuje.

### 6.3 Asinhronska serijska komunikacija (SCI)

Za asinhrono komunikacijo s periferijo (npr. s terminalom ali z nadrejenim računalnikom - navadno PC) uporabljamo SCI modul (SCI je prisoten tudi pri drugih Motorolinih mikrokrmlnikih, npr. MC68HC11 in MC68HC05). Nekatere značilnosti SCI so:

- Izbira prenosa 8- ali 9-bitne besede.
- Detekcija motenj.
- Generiranje sode ali lihe (angl. even/odd) paritete.
- Programirljiva hitrost (frekvanca) prenosa itd.

#### 6.3.1 Serijska komunikacija med mikroprocesorjem in nadrejenim računalnikom

Preden si ogledamo osnovo delovanja SCI modula, povejmo najprej nekaj o njegovi najpogosteji uporabi: serijski komunikaciji z nadrejenim računalnikom (kot npr. v standardni konfiguraciji BCC, glej poglavje 8), ki običajno poteka preko vmesnika EIA-232C (prej RS-232C, znan tudi kot V.24).

Znotraj mikroprocesorja (mikrokrmlnika) tečejo podatki po paralelnem podatkovnem vodilu (npr. 8-, 16- ali 32-bitnem), komunikacija s terminalom pa je asinhronska serijska<sup>8</sup>. Za usklajevanje teh dveh načinov prenosa podatkov rabimo posebno vmesno prilagodilno vezje UART (Universal Asynchronous Receiver/Transmitter), pri nakaterih mikrokrmlnikih pa je UART že njihov sestavni modul (npr. SCI v

<sup>8</sup> Najosnovnejša komunikacija je možna le s tremi vodniki: TxD, RxD in maso.

MC68332). UART je sestavljen iz serijsko/paralelnega in paralelno serijskega vmesnika, ki imata neodvisna dajalnika takta. Serijska komunikacija je dvosmerna:

1. UART pošilja terminalu ali osebnem računalniku podatke prek nožice TxD (angl. Transmit Data - pošiljanje podatkov). Običajno se prek nje izvršuje prikaz podatkov z mikroprocesorja na zaslon terminala ali prenos nekaterih internih parametrov na diskovno enoto (npr. del vsebine pomnilnika).
2. V obratno smer (nožica RxD - angl. Receive Data) sprejema UART podatke s terminala ali z nadrejenega računalnika.

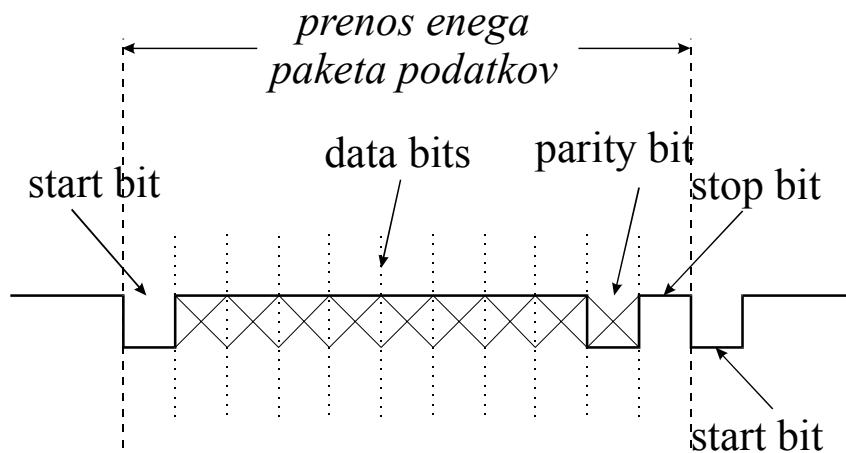
EIA-232 standard je primer “negativne logike”, ki smo jo omenili v poglavju 1. Logičnemu signalu 0 na strani oddajnika ustreza napetostni pas (+5 V, +15 V), signalu 1 pa (-15 V, -5 V), na strani sprejemnika pa sta pasova (+3 V, +15 V) in (-15 V, -3 V) [31]. Po drugi strani sta običajna nivoja na strani mikroprocesorja 0 V in 5 V (TTL). Zato moramo načeloma med obema enotama izvršiti ustrezno prilagoditev nivojev s posebnimi vezji (pogl. 12.3).

Glede na asinhronski način komunikacije so “hitrosti” pošiljanja signalov, ki se meri s t.i. številom *bitov na sekundo*, standardno določene. Običajne vrednosti so 1200, 2400 (pri počasnejših prenosih, npr. pri starejših modemih), 9600 in 19200 bps (standardni hitrosti pri komunikaciji s terminalom) itd.

Na obeh linijah je pred začetkom prenosa prisoten logični signal 1 (glej sliko 6. 12). Asinhronski transfer paketa signalov se začne, ko pošlje oddajnik po svoji TxD liniji t.i. *start bit* (logični nivo 0). Startnemu bitu sledi sedem (kompatibilnost z ASCII kodo) ali osem *podatkovnih bitov* po prej določeni hitrosti.

Za njimi sledi *parnostni bit* (angl. parity bit). Pri prenosu je namreč možno popačenje stanja bita kot posledica motenj. Najenostavnejši način preverjanja pravilnosti prenosa je uporaba posebnega bita, ki se setira, če število prenešenih bitov ustreza predhodno določenemu tipu paritete: liha ali soda. Preverjanje paritete lahko tudi razveljavimo. Če npr. izberemo sodo paritet, se paritetni bit setira pri sodem številu enic v serijskem paketu oz. resetira, če je število enic liho. Obratno velja, če izberemo liho paritet. Seveda detektira takšna kontrola le izgubo lihega števila bitov.

Prenos se konča z enim ali dvema *stop bitoma*.



*Slika 6. 12: Format asinhronskega serijskega paketa podatkov*

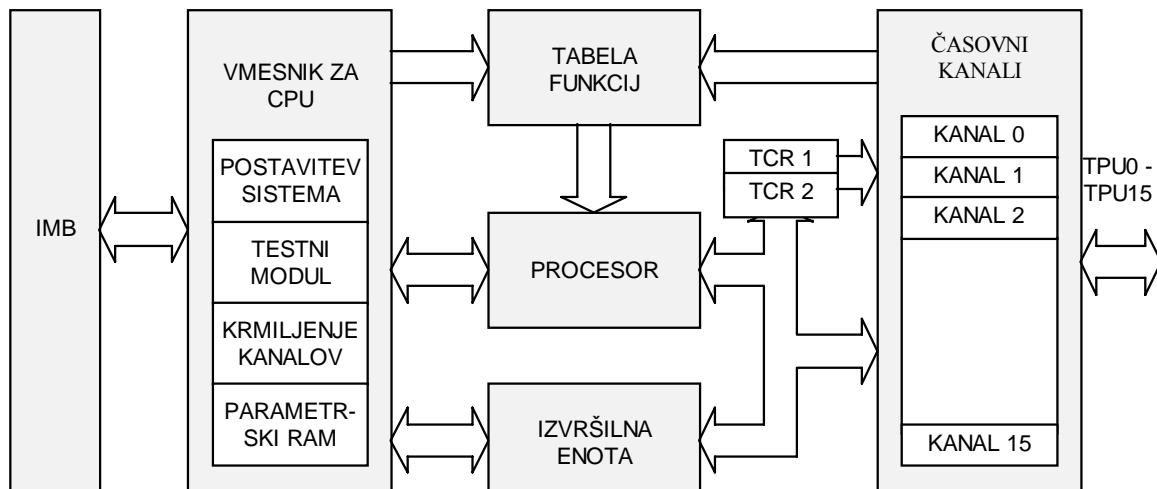
Vsi do sedaj opisani parametri (hitrost prenosa, število podatkovnih bitov itd.) morajo biti definirani na obeh straneh komunikacijske verige. Pri mikrokrmlniku jih, podobno kot pri QSPI, definiramo v posebnih registrih SCI (SCCR0, SCCR1, SCSR in SCDR, slika 6. 2).

Te parametre je treba uskladiti s komunikacijskimi parametri terminala ali osebnega računalnika. Temu je namenjena posebna programska oprema, ki koordinira komunikacijo na tej strani komunikacijskega kanala. Pri navadnem terminalu (v bistvu le tipkovnica in zaslon) shranimo te parametre v posebno spominsko področje.

PC lahko tudi deluje kot terminal. Za to uporabljamo standardne programske pakete (npr. starejše Kermit, Telnet, ProCom, VTerm v DOS ali novejše, kot je npr. HyperTerminal v Windows okolju).

## 7. ČASOVNO-PROCESNA ENOTA - TPU

Mikrokrmlnik MC68332 vsebuje poleg CPU-ja še dodatni procesor, t.i. *časovno-procesno enoto* (angl. Time Processing Unit - TPU). Zato govorimo o multiprocesorski konfiguraciji. TPU je zadolžena za zajemanje in izdajanje binarnih vrednosti (1 in 0) v dosvisnosti od časa. Slika 7. 1 kaže blokovno shemo TPU-ja (glej tudi pogl. 3).



*Slika 7. 1: Blokovna shema časovno-procesne enote*

TPU vsebuje nekatere vnaprej programirane časovne funkcije, ki jih je za njihovo uporabo treba ustrezno parametrirati<sup>1</sup>. Večino teh funkcij lahko uporabimo na kateremkoli izmed šestnajstih kanalov. Funkcije razdelimo na dve skupini:

- **vhodne časovne funkcije** so predvidene za zajemanje in ovrednotenje binarnih časovnih signalov,
- za izdajanje vnaprej definirane sekvence pulzov uporabljam **izhodne časovne funkcije**.

Z ozirom na že programirane TPU funkcije imamo na voljo dve enačici mikrokrmlnika MC68332: A in G. Pri prvi vsebuje TPU splošnejše funkcije, druga pa je prilagojena krmiljenju in regulaciji elektromotorjev. Nekatere izmed časovnih funkcij so:

- **Zajemanje vhodnih signalov oz. štetje prehodov na vhodu** (angl. Input Capture/Input Transition Counter - ITC) - (inačica A). S funkcijo ITC merimo čas med izbranimi prehodi vhodnega signala.

<sup>1</sup> Časovne funkcije lahko prilagodimo našim posebnim zahtevam, za kar pa je potrebno dodatno poznavanje najnižjega nivoja programiranja: *mikroprogramiranja*. Vsak ukaz (npr. CPU ukaz) je sestavljen iz sekvence še enostavnejših ukazov. Delo z mikroprogramom je izredno zahtevno in uporabniku načeloma onemogočeno, zato ga ne bomo posebej obravnavali.

- **Diskretni vhodi/izhodi** (angl. Discrete Input/Output) - (A). Če izberemo vhodno funkcijo, se v posebnem registru shranjuje stanje prehodov v zadnjih šestnajstih urinih ciklih TPU. Pri izhodni funkciji pa se na izhod pošlje vnaprej definirana sekvenca stanj.
- **Primerjanje vhodov** (angl. Output Compare) - (A). Generiranje stanja 1, 0 ali sprememba prejšnjega stanja (angl. toggle) z določeno zakasnitvijo.
- **Pulzno-širinska modulacija** (angl. Pulse-Width Modulation) - (A) je zelo pogosto uporabljana funkcija, o kateri bomo več povedali v nadaljevanju (pogl. 10.3.5.7). Z njo lahko določimo vlak pulzov, ki jim lahko spremojamo periodo in trajanje stanja 1.
- **Koračni motor** (angl. Stepper Motor) - (A). Vrtenje koračnih motorjev dosežemo z generiranjem dveh vlakov pulzov, ki sta premaknjena za  $90^\circ$ , na statorskih navitjih. Smer vrtenja je odvisna od zaporedja faz (pogl. 12.6). Funkcija omogoča programirano pospeševanje, zaviranje in obratovalno stanje, kjer je trenutna hitrost motorja proporcionalna frekvenci vhodnih pulzov.
- **Akumulacija trajanja periode oz. širine pulzov** (angl. Period/Pulse-Width Accumulator - PPWA) - (A). Funkcija omogoča merjenje skupnega trajanja izbranega števila period ali skupno trajanje pulzov v določenem številu period.
- **Zajemanje dveh časovno premaknjenih pulzov** (angl. Quadrature Decode - QDEC) - (A) uporabimo pri ugotavljanju kota zasuka rotorja z inkrementalnim dajalnikom (pogl. 12.5.2).
- **Merjenje frekvence** (angl. Frequency Measurement - FQM) - (G). Znotraj vnaprej določenega časovnega intervala štejemo periode.
- **Asinhronski oddajnik/sprejemnik** (angl. Universal Asynchronous Receiver/Transmitter - UART) - (G) smo že spoznali v poglavju 6.3 o SCI modulu. Možna je uporaba osmih UART-ov.
- **Večfazna komutacija** (angl. Multiphase Motor Commutation - COMM) - (G) omogoča generiranje sekvenč za krmiljenje enosmernih elektronsko komutiranih motorjev (angl. brushless motors). Funkcijo lahko uporabimo tudi na trifaznih motorjih.
- Pogoj za pravilno delovanje elektronsko komutiranih motorjev (glej prejšnjo funkcijo) je preklapljanje faz sinhrono s kotom rotorja oz. s položajem trajnega magneta, za kar so zadolženi Hallovi senzorji. V ta namen uporabljamо **funkcijo dekodiranja signala iz Hallovin senzorjev** (angl. Hall Effect Decode - HALLD) - (G), ki deluje skupaj s funkcijo COMM. Možno je dekodiranje signalov iz dveh ali treh senzorjev.

O eni izmed funkcij (PWM) in načinu parametriranja TPU-ja bomo več govorili v nadaljevanju. Na tem mestu omenimo osnovno prednost posebnega časovnega procesorja:

- Funkcije TPU bistveno zmanjšajo potrebo po aparurni opremi za zajemanje, obdelavo in generiranje časovnih pulzov (npr. posebni števci, flip-flopi itd.).
- TPU deluje povsem neodvisno od CPU, s katero komunicira le preko prekinitev. Na ta način se občutno zmanjša potreba po posegih CPU pri obdelavi časovnih funkcij.

TPU vsebuje 16 kanalov (angl. channels) oz. priključnih sponk, ki jih označujemo z oznakami TPU0 - TPU15 (glej slike 7. 1 in 3. 3). Na vsakem od teh kanalov lahko uporabimo poljubno časovno funkcijo. Kanali lahko obratujejo samostojno ali v povezavi z drugimi kanali (npr. pri generiranju krmilnih pulzov za koračni motor sodelujeta dva kanala, ki sta konfigurirana kot izhoda).

## 7.1 Registri in RAM pomnilnik TPU modula

V prejšnjih poglavjih smo povedali, da so nekatera področja v pomnilniški mapi MC68332 rezervirana za registre in pomnilnike podmodulov. Za TPU je rezervirano področje  $Yffe00_{HEX}$  -  $Yffeff_{HEX}$ <sup>2</sup>. V njem se nahajata:

- **registrska mapa in**
- **parametrski RAM.**

Na omenjenih naslovih se mora nahajati RAM pomnilnik ker moramo imeti možnost sprotnega spreminjanja vsebin obeh področij.

### 7.1.1 Registrska mapa TPU-ja

Razlikujemo tri vrste TPU registrov:

- **Registri za sistemsko konfiguracijo,**
- **statusni registri in registri za krmiljenje kanalov ter**
- **registri za podporo razvoja in preizkušanje.**

Osnovne funkcije registrov so:

- določanje osnovne konfiguracije celotnega TPU-ja (osnovni takt delovanja, prekinitvena prioriteta in ustrezni vektorji...) ter
- konfiguriranje posameznih kanalov (izbira časovne funkcije in njenih parametrov, dodelitev in spremljanje izvajanja prekinitev...).

Registrska mapa TPU-ja kaže slika 7. 2.

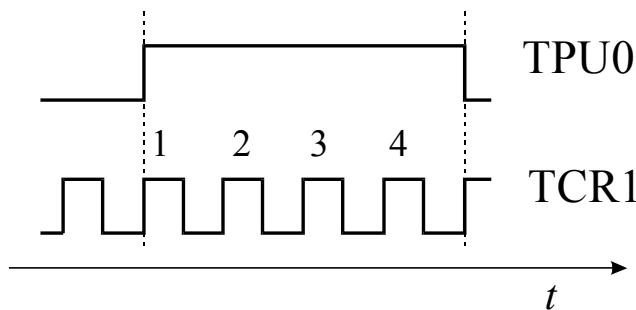
Vsi uporabljeni kanali oz. njihove funkcije so sinhronizirani na enega izmed dveh 16-bitnih prostotekočih časovnikov TCR1 in TCR2 (angl. Timer Count Registers - registri za štetje časa). Frekvenci obeh moramo nastaviti pri inicjalizaciji TPU. Dolžina periode TCR1 je mnogokratnik dolžine periode osnovnega dajalnika takta MC68332 (v našem primeru obravnavamo le inačico s frekvenco 16,77 MHz, to je s periodom 59,6 ns), za določanje frekvence TCR2 pa lahko uporabimo še nek dodatni zunanji dajalnik takta (glej poglavje 7.1.1.1). Na sliki 7. 3 je prikazan primer generiranja pulza (signal 1) na kanalu TPU0 v trajanju štirih period TCR1 (funkcija PWM).

---

<sup>2</sup> Y je  $f_{HEX}$  ali  $7_{HEX}$ , odvisno od izbire bita MM v SIM registru MCR (naslov  $Yffa00_{HEX}$ ). Običajno velja  $Y = f_{HEX}$ .

| Naslov:               | 15 | BYTE n | 8   7 | BYTE n+1                                    | 0 |
|-----------------------|----|--------|-------|---|---|
| Yffe00 <sub>HEX</sub> |    |        |       | Module Configuration Register (TPUMCR)      |   |
| Yffe02 <sub>HEX</sub> |    |        |       | Test Configuration Register                 |   |
| Yffe04 <sub>HEX</sub> |    |        |       | Development Support Control Register        |   |
| Yffe06 <sub>HEX</sub> |    |        |       | Development Support Status Register         |   |
| Yffe08 <sub>HEX</sub> |    |        |       | Interrupt Configuration Register (TICR)     |   |
| Yffe0a <sub>HEX</sub> |    |        |       | Channel Interrupt Enable Register (CIER)    |   |
| Yffe0c <sub>HEX</sub> |    |        |       | Channel Function Select Register 0 (CFSR 0) |   |
| Yffe0e <sub>HEX</sub> |    |        |       | Channel Function Select Register 1 (CFSR 1) |   |
| Yffe10 <sub>HEX</sub> |    |        |       | Channel Function Select Register 2 (CFSR 2) |   |
| Yffe12 <sub>HEX</sub> |    |        |       | Channel Function Select Register 3 (CFSR 3) |   |
| Yffe14 <sub>HEX</sub> |    |        |       | Host Sequence Register 0 (HSQR 0)           |   |
| Yffe16 <sub>HEX</sub> |    |        |       | Host Sequence Register 0 (HSQR 1)           |   |
| Yffe18 <sub>HEX</sub> |    |        |       | Host Service Request Register 0 (HSRR 0)    |   |
| Yffe1a <sub>HEX</sub> |    |        |       | Host Service Request Register 1 (HSRR 1)    |   |
| Yffe1c <sub>HEX</sub> |    |        |       | Channel Priority Register 0 (CPR 0)         |   |
| Yffe1e <sub>HEX</sub> |    |        |       | Channel Priority Register 1 (CPR 1)         |   |
| Yffe20 <sub>HEX</sub> |    |        |       | Channel Interrupt Status Register (CISR)    |   |
| Yffe22 <sub>HEX</sub> |    |        |       | Link Register                               |   |
| Yffe24 <sub>HEX</sub> |    |        |       | Service Grant Latch Register                |   |
| Yffe26 <sub>HEX</sub> |    |        |       | Decoded Channel Number Register             |   |
| Yffe28 <sub>HEX</sub> |    |        |       |   |   |
| .....                 |    |        |       |   |   |
| Yffeef <sub>HEX</sub> |    |        |       | ZASEDENO                                    |   |

Slika 7. 2: Registrska mapa TPU-ja



Slika 7. 3: Primer sinhronizacije časovnega signala s TCR1 ali TCR2

Delo s TPU-jem poteka v treh korakih:

1. Inicializacija: nastavitev osnovne konfiguracije TPU, definiranje uporabljenih kanalov in njihovih funkcij, zagon TPU.
2. Izvajanje TPU funkcij neodvisno od CPU-ja: TCR1 ali TCR2 obratujeta kot prostotekoča dajalnika takta.
3. Prekinitve delovanja TPU - možna le ob resetiranju celotnega MC68332 oz. ob postaviti bita STOP v registru TPUMCR. Drugače izvajanja funkcij TPU ne moremo prekiniti, celo z ABORT prekinivijo ne.

Med obratovanjem TPU-ja neodvisno od izvajanja CPU instrukcij (točka 2) lahko obe procesorski enoti komunicirata (CPU je nadrejena). Pri vseh funkcijah je možno generiranje zahteve po prekinitvi delovanja CPU-ja s strani TPU-ja. Npr. takšno zahtevo je možno generirati na koncu vsake periode TPU funkcije PWM. Ali se bo CPU na zahtevo odzval, je odvisno od nastavitev v fazi inicializacije (glej TPU registra TICR in CIER v nadaljevanju).

V naslednjih podpoglavljih si bomo ogledali registre TPU-ja ter nekatere njihove funkcije.

#### 7.1.1.1 Register za konfiguracijo TPU modula (TPUMCR)

| TPUMCR |                |                |     |      |     |      |      |   |   |   |   |   |   |   |   | Naslov: Yffe00 <sub>HEX</sub> |   |
|--------|----------------|----------------|-----|------|-----|------|------|---|---|---|---|---|---|---|---|-------------------------------|---|
| 15     | 14             | 13             | 12  | 11   | 10  | 9    | 8    | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |                               |   |
| STOP   | TCR1<br>PRESC. | TCR2<br>PRESC. | EMU | T2CG | STF | SUPV | PSCK | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ARBITRAŽA PREKINITEV<br>IARB  |   |
| RES:   |                |                |     |      |     |      |      |   |   |   |   |   |   |   |   |                               |   |
| 0      | 0              | 0              | 0   | 0    | 0   | 0    | 0    | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0                             | 0 |

#### **STOP - Stop bit**

Setiranje tega bita pomeni ustavitev delovanja TPU-ja, kar signalizira bit STF.

#### **TCR1 presc. (TCR1 Prescaler Control)**

#### **PSCK - Prescaler Clock**

Ti trije biti določajo frekvenco inkrementiranja časovnika TCR1 po naslednji tabeli (slika 7.4):

| TCR1 presc. | PSCK = 0   |                               | PSCK = 1   |                               |
|-------------|------------|-------------------------------|------------|-------------------------------|
|             | Št. period | Dolž. periode<br>(pri 16 MHz) | Št. period | Dolž. periode<br>(pri 16 MHz) |
| 00          | 32         | ≈ 2 µs                        | 4          | ≈ 250 ns                      |
| 01          | 64         | ≈ 4 µs                        | 8          | ≈ 500 ns                      |
| 10          | 128        | ≈ 8 µs                        | 16         | ≈ 1 µs                        |
| 11          | 256        | ≈ 16 µs                       | 32         | ≈ 2 µs                        |

*Slika 7.4: Nastavljanje frekvence TCR1 v odvisnosti od bitov TCR1 presc. in PSCK*

Podoben princip velja tudi za polja TCR2 presc.

#### **IARB biti - Interrupt Arbitration**

Vsak modul v MC68332, ki je povezan na intermodularni bus (IMB), lahko generira prekinitve delovanja CPU-ja. IARB polje določa prioriteto modula (tukaj TPU-ja), ki je potrebna za arbitražo, ko pride do hkratne zahteve po prekinitvi s strani več modulov.

### 7.1.1.2 Register za konfiguracijo prekinitiv TPU-ja (TICR)

TICR (TPU Interrupt Configuration Register) vsebuje le dve polji: nivo zahteve po prekinitvi kanalov (angl. Channel Interrupt Request Level) in bazni prekinitveni vektor kanalov (angl. Channel Interrupt Base Vector).

| TICR |    |    |    |    |                                  |   |   |                                  |   |   |   |   |   |   |   | Naslov: YFFE08 <sub>HEX</sub> |   |   |   |
|------|----|----|----|----|----------------------------------|---|---|----------------------------------|---|---|---|---|---|---|---|-------------------------------|---|---|---|
| 15   | 14 | 13 | 12 | 11 | 10                               | 9 | 8 | 7                                | 6 | 5 | 4 | 3 | 2 | 1 | 0 |                               |   |   |   |
| 0    | 0  | 0  | 0  | 0  | CHANNEL INTERR.<br>REQUEST LEVEL |   |   | CHANNEL INTERRUPT<br>BASE VECTOR |   |   | 0 | 0 | 0 | 0 | 0 | 0                             | 0 | 0 | 0 |
| RES: | 0  | 0  | 0  | 0  | 0                                | 0 | 0 | 0                                | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0                             | 0 | 0 | 0 |

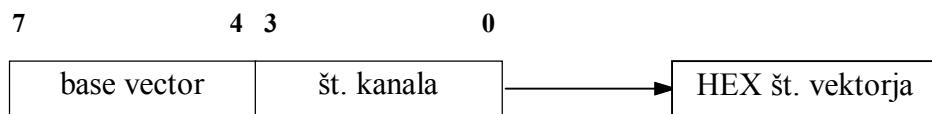
V poglavju o prekinitvah smo povedali, da obstaja veliko razlogov za generiranje prekinitiv (npr. softverska: deljenje z ničlo; hardverska: zunanji signal). Zaradi izvajanja ustreznega podprograma je treba ugotoviti kaj je vzrok za prekinitiv. Prekinitveni vektorji so shranjeni v posebnem področju RAM-a. Vektorji vsebujejo začetne naslove prekinitvenih podprogramov in morajo biti definirani pred zagonom glavnega programa (med inicializacijo). Večina vektorjev ustreza točno določenemu tipu prekinitve, nekatere vektorje pa lahko dodelimo poljubnim prekinitvam.

Vsek vektor vsebuje svojo zaporedno številko; njegov dejanski naslov izračunamo z izrazom:

$$\text{zaporedna številka vektorja} \cdot 4 + (\text{VBR}).$$

Spomnimo se: CPU register VBR (Vector Base Register) vsebuje lokacijo ničtega vektorja v RAM pomnilniku. Zaporedno številko vektorja množimo s štiri, saj je vsak vektor sestavljen iz štirih bytov ali ene dolge besede (le na ta način lahko zapišemo 24-bitni naslov).

Prekinitve, ki jih generirajo TPU funkcije na posameznih kanalih, sodijo med tiste, katerih vektorji nimajo vnaprej določenih pozicij v vektorski mapi, zato jim moramo pred zagonom podati ustrezone naslove. Zaporedna številka vektorja je določena z vsebino polja Channel Interrupt Base Vector v TICR registru (zgornji nibble številke vektorja) in številko TPU kanala (spodnji nibble - od 0<sub>HEX</sub> do f<sub>HEX</sub>):



Postavljanje prekinitvenega vektorja je opisano v poglavju 7.2.

Prioritetno TPU prekinitive določimo v polju Channel Interrupt Request Level, kjer ima prekinitiv z najvišjo prioriteto zaporedno številko sedem (vsi biti setirani).

Do sedaj opisana registra vsebuje splošne parametre TPU. V naslednji skupini registrov pa so splošni parametri za posamezne kanale.

### 7.1.1.3 Register za omogočanje prekinitve kanalov (CIER)

Časovne funkcije na posameznih kanalih lahko generirajo prekinitve le, če je v CIER registru (angl. Channel Interrupt Enable Register) setiran pripadajoči biti.

CIER

Naslov: Yffe0a<sub>HEX</sub>

| 15       | 14       | 13       | 12       | 11       | 10       | 9       | 8       | 7       | 6       | 5       | 4       | 3       | 2       | 1       | 0       |
|----------|----------|----------|----------|----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| kanal 15 | kanal 14 | kanal 13 | kanal 12 | kanal 11 | kanal 10 | kanal 9 | kanal 8 | kanal 7 | kanal 6 | kanal 5 | kanal 4 | kanal 3 | kanal 2 | kanal 1 | kanal 0 |
| RES:     |          |          |          |          |          |         |         |         |         |         |         |         |         |         |         |
| 0        | 0        | 0        | 0        | 0        | 0        | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |

### 7.1.1.4 Statusni register prekinitve kanalov (CISR)

Če na določenem TPU kanalu pride do prekinitve, se to označi s stanjem 1 v pripadajočem bitu registra CISR (angl. Channel Interrupt Status Register).

CISR

Naslov: Yffe20<sub>HEX</sub>

| 15       | 14       | 13       | 12       | 11       | 10       | 9       | 8       | 7       | 6       | 5       | 4       | 3       | 2       | 1       | 0       |
|----------|----------|----------|----------|----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| kanal 15 | kanal 14 | kanal 13 | kanal 12 | kanal 11 | kanal 10 | kanal 9 | kanal 8 | kanal 7 | kanal 6 | kanal 5 | kanal 4 | kanal 3 | kanal 2 | kanal 1 | kanal 0 |
| RES:     |          |          |          |          |          |         |         |         |         |         |         |         |         |         |         |
| 0        | 0        | 0        | 0        | 0        | 0        | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       | 0       |

### 7.1.1.5 Registri za izbiro časovnih funkcij kanalov (CFSR0, CFSR1, CFSR2, CFSR3)

Z naslednjimi štirimi CFSR registri (angl. Channel Function Select Registers) izberemo časovno funkcijo za posamezne kanale.

CFSR0

Naslov: Yffe0c<sub>HEX</sub>

| 15       | 14 | 13 | 12 | 11       | 10 | 9 | 8 | 7        | 6 | 5 | 4 | 3        | 2 | 1 | 0 |
|----------|----|----|----|----------|----|---|---|----------|---|---|---|----------|---|---|---|
| KANAL 15 |    |    |    | KANAL 14 |    |   |   | KANAL 13 |   |   |   | KANAL 12 |   |   |   |
| RES:     |    |    |    |          |    |   |   |          |   |   |   |          |   |   |   |
| 0        | 0  | 0  | 0  | 0        | 0  | 0 | 0 | 0        | 0 | 0 | 0 | 0        | 0 | 0 | 0 |

CFSR1

Naslov: Yffe0e<sub>HEX</sub>

| 15       | 14 | 13 | 12 | 11       | 10 | 9 | 8 | 7       | 6 | 5 | 4 | 3       | 2 | 1 | 0 |
|----------|----|----|----|----------|----|---|---|---------|---|---|---|---------|---|---|---|
| KANAL 11 |    |    |    | KANAL 10 |    |   |   | KANAL 9 |   |   |   | KANAL 8 |   |   |   |
| RES:     |    |    |    |          |    |   |   |         |   |   |   |         |   |   |   |
| 0        | 0  | 0  | 0  | 0        | 0  | 0 | 0 | 0       | 0 | 0 | 0 | 0       | 0 | 0 | 0 |

CFSR2

Naslov: Yffe10<sub>HEX</sub>

| 15      | 14 | 13 | 12 | 11      | 10 | 9 | 8 | 7       | 6 | 5 | 4 | 3       | 2 | 1 | 0 |
|---------|----|----|----|---------|----|---|---|---------|---|---|---|---------|---|---|---|
| KANAL 7 |    |    |    | KANAL 6 |    |   |   | KANAL 5 |   |   |   | KANAL 4 |   |   |   |
| RES:    |    |    |    |         |    |   |   |         |   |   |   |         |   |   |   |
| 0       | 0  | 0  | 0  | 0       | 0  | 0 | 0 | 0       | 0 | 0 | 0 | 0       | 0 | 0 | 0 |

| CFSR3   |    |    |    |         |    |   |   |         |   |   |   |         |   |   |   | Naslov: Yffe12 <sub>HEX</sub> |   |  |  |
|---------|----|----|----|---------|----|---|---|---------|---|---|---|---------|---|---|---|-------------------------------|---|--|--|
| 15      | 14 | 13 | 12 | 11      | 10 | 9 | 8 | 7       | 6 | 5 | 4 | 3       | 2 | 1 | 0 |                               |   |  |  |
| KANAL 3 |    |    |    | KANAL 2 |    |   |   | KANAL 1 |   |   |   | KANAL 0 |   |   |   |                               |   |  |  |
| RES:    |    |    |    |         |    |   |   |         |   |   |   |         |   |   |   |                               |   |  |  |
| 0       | 0  | 0  | 0  | 0       | 0  | 0 | 0 | 0       | 0 | 0 | 0 | 0       | 0 | 0 | 0 | 0                             | 0 |  |  |

Tabela na sliki 7. 5 kaže bitne vzorce oz. kode nekaterih funkcij. V tabeli so zbrane tudi opcije posameznih funkcij, ki jih definirajo registri HSQR in HSRR (opisani v nadaljevanju). Vrednost 0<sub>HEX</sub> v polju "koda funkcije" pomeni, da kanalu ni dodeljena nobena funkcija (kanal je neaktivен).

| Naziv funkcije (kratica) | Hex koda funkcije pogl. 7.1.1.5 | Koda HSRR pogl. 7.1.1.7   | Koda HSQR pogl. 7.1.1.6  |
|--------------------------|---------------------------------|---|--|
| PPWA                     | f                               | 0 = nič<br>1 = neuporabljeno<br>2 = inicializacija<br>3 = neuporabljeno   | 0 = 24-bitna perioda<br>1 = 16-bitna perioda+povezava<br>2 = 24-bitna širina pulza<br>3 = 16-bitna širina pulza+povezava   |
| DIO                      | 8                               | 0 = nič<br>1 = postavi izhod v stanje 1<br><br>2 = postavi izhod v stanje 0<br><br>3 = inicializacija, vhod določen<br><br>3 = inicializacija, periodični vhod<br><br>3 = ažuriraj parameter statusa nožice | 0<br>0 = Zajemanje sprememb: shrani stanje nožice ob sprememb<br>0 = Zajemanje sprememb: shrani stanje nožice ob sprememb<br>0 = Zajemanje sprememb: shrani stanje nožice ob sprememb<br>1 = Ujemanje: shrani stanje nožice pri MATCH_RATE<br>2 = Shrani stanje nožice v HSR11 |
| ITC                      | a                               | 0 = nič<br>1 = inicializacija<br>2,3 = neuporabljeno  | 0 = brez povezav, eno zajemanje<br>1 = brez povezav, stalno zajemanje<br>2 = povezava z drugimi kanali, eno zajemanje<br>3 = povezava z drugimi kanali, stalno zajemanje   |
| PWM                      | 9                               | 0 = nič<br>1 = signalizacija zahteve po ažuriranju<br>2 = inicializacija<br>3 = neuporabljeno   | neuporabljeno  |
| SM                       | d                               | 0 = nič<br>1 = nič<br>2 = inicializacija<br>3 = zahtev po koraku  | neuporabljeno  |

**Slika 7. 5: Izbiranje nekaterih funkcij in njihovih opcij**

#### 7.1.1.6 Registra za sekvence CPU-ja (HSQR0 in HSQR1)

HSQR0

Naslov: Yffe14<sub>HEX</sub>

|             |             |             |             |             |             |            |            |   |   |   |   |   |   |   |   |
|-------------|-------------|-------------|-------------|-------------|-------------|------------|------------|---|---|---|---|---|---|---|---|
| 15          | 14          | 13          | 12          | 11          | 10          | 9          | 8          | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Kanal<br>15 | Kanal<br>14 | Kanal<br>13 | Kanal<br>12 | Kanal<br>11 | Kanal<br>10 | Kanal<br>9 | Kanal<br>8 |   |   |   |   |   |   |   |   |
| RES:        |             |             |             |             |             |            |            |   |   |   |   |   |   |   |   |
| 0           | 0           | 0           | 0           | 0           | 0           | 0          | 0          | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

HSQR1

Naslov: Yffe16<sub>HEX</sub>

|            |            |            |            |            |            |            |            |   |   |   |   |   |   |   |   |
|------------|------------|------------|------------|------------|------------|------------|------------|---|---|---|---|---|---|---|---|
| 15         | 14         | 13         | 12         | 11         | 10         | 9          | 8          | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Kanal<br>7 | Kanal<br>6 | Kanal<br>5 | Kanal<br>4 | Kanal<br>3 | Kanal<br>2 | Kanal<br>1 | Kanal<br>0 |   |   |   |   |   |   |   |   |
| RES:       |            |            |            |            |            |            |            |   |   |   |   |   |   |   |   |
| 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Registra sta namenjena izbiri načina obratovanja izbrane časovne funkcije. Dostop do HSQR (angl. Host Sequence Register) ima vodilni modul na IMB (angl. master ali host), običajno pa je to kar CPU. Pomen bitov je odvisen od izbrane časovne funkcije, kot je to prikazano na sliki 7. 5.

#### 7.1.1.7 Registra za zahtevo po servisiranju CPU-ja (HSRR0 in HSRR1)

HSRR0

Naslov: Yffe18<sub>HEX</sub>

|             |             |             |             |             |             |            |            |   |   |   |   |   |   |   |   |
|-------------|-------------|-------------|-------------|-------------|-------------|------------|------------|---|---|---|---|---|---|---|---|
| 15          | 14          | 13          | 12          | 11          | 10          | 9          | 8          | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Kanal<br>15 | Kanal<br>14 | Kanal<br>13 | Kanal<br>12 | Kanal<br>11 | Kanal<br>10 | Kanal<br>9 | Kanal<br>8 |   |   |   |   |   |   |   |   |
| RES:        |             |             |             |             |             |            |            |   |   |   |   |   |   |   |   |
| 0           | 0           | 0           | 0           | 0           | 0           | 0          | 0          | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

HSRR1

Naslov: Yffe1a<sub>HEX</sub>

|            |            |            |            |            |            |            |            |   |   |   |   |   |   |   |   |
|------------|------------|------------|------------|------------|------------|------------|------------|---|---|---|---|---|---|---|---|
| 15         | 14         | 13         | 12         | 11         | 10         | 9          | 8          | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Kanal<br>7 | Kanal<br>6 | Kanal<br>5 | Kanal<br>4 | Kanal<br>3 | Kanal<br>2 | Kanal<br>1 | Kanal<br>0 |   |   |   |   |   |   |   |   |
| RES:       |            |            |            |            |            |            |            |   |   |   |   |   |   |   |   |
| 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

HSRR registra (angl. Host Service Request Registers) določata eno izmed treh možnih interakcij med določenim TPU kanalom in CPU-jem oz. nekim drugim nadrejenim modulom IMB vodila (slika 7. 5). Četrta možna kombinacija bitov (00<sub>BIN</sub>) je začetno stanje, ki se ponovno vzpostavi po servisiranju kanala, zato tudi označuje konec izvajanja postopka, ki je bil izbran z eno izmed ostalih treh kombinacij.

#### 7.1.1.8 Registra za določanje prioritete kanalov (CPR0 in CPR1)

CPR0

Naslov: Yffe1c<sub>HEX</sub>

|             |             |             |             |             |             |            |            |   |   |   |   |   |   |   |   |
|-------------|-------------|-------------|-------------|-------------|-------------|------------|------------|---|---|---|---|---|---|---|---|
| 15          | 14          | 13          | 12          | 11          | 10          | 9          | 8          | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Kanal<br>15 | Kanal<br>14 | Kanal<br>13 | Kanal<br>12 | Kanal<br>11 | Kanal<br>10 | Kanal<br>9 | Kanal<br>8 |   |   |   |   |   |   |   |   |
| RES:        |             |             |             |             |             |            |            |   |   |   |   |   |   |   |   |
| 0           | 0           | 0           | 0           | 0           | 0           | 0          | 0          | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

CPR1

Naslov: Yffe1e<sub>HEX</sub>

|            |            |            |            |            |            |            |            |   |   |   |   |   |   |   |   |
|------------|------------|------------|------------|------------|------------|------------|------------|---|---|---|---|---|---|---|---|
| 15         | 14         | 13         | 12         | 11         | 10         | 9          | 8          | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Kanal<br>7 | Kanal<br>6 | Kanal<br>5 | Kanal<br>4 | Kanal<br>3 | Kanal<br>2 | Kanal<br>1 | Kanal<br>0 |   |   |   |   |   |   |   |   |
| RES:       |            |            |            |            |            |            |            |   |   |   |   |   |   |   |   |
| 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0          | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Za določanje prioritete kanala oz. verjetnosti servisiranja sta zadolžena CPR registra (angl. Channel Priority Registers):

| Bit 1 | Bit 2 | Prioriteta servisiranja                           |
|-------|-------|---|
| 0     | 0     | kanal zapahnjen<br>(angl. disabled) - zač. stanje |
| 0     | 1     | nizka   |
| 1     | 0     | srednja   |
| 1     | 1     | visoka  |

#### 7.1.2 Parametrski RAM TPU-ja

Po splošnem definirjanju načina delovanja TPU-ja in izbiri časovnih funkcij na posameznih kanalih sledi določanje parametrov, ki jih zahtevajo funkcije na izbranih kanalih. V ta namen je v TPU RAM-u rezervirano 256 bytov, ki so razdeljeni na šestnajst skupin (za TPU0 - TPU15) po osem besed (za osem različnih parametrov - slika 7. 6). Tako bo npr. prvi parameter funkcije, ki smo jo dodelili TPU8, na lokaciji Yfff80<sub>HEX</sub>, drugi na Yfff82<sub>HEX</sub> itd. Besedi 6 in 7 se le poredkoma uporabljata in sta na voljo samo pri kanalih TPU14 in TPU15.

| TPU<br>kanal: | Registri parametrskega RAM-a |                        |                        |                        |                        |                                   |                        |                        |
|---------------|------------------------------|------------------------|------------------------|------------------------|------------------------|-----------------------------------|------------------------|------------------------|
|               | 0                            | 1                      | 2                      | 3                      | 4                      | 5                                 | 6                      | 7                      |
| 0             | Yfff00 <sub>HEX</sub>        | Yfff02 <sub>HEX</sub>  | Yfff04 <sub>HEX</sub>  | Yfff06 <sub>HEX</sub>  | Yfff08 <sub>HEX</sub>  | Yfff0a <sub>HEX</sub>             | -                      | -                      |
| 1             | Yfff10 <sub>HEX</sub>        | Yfff12 <sub>HEX</sub>  | Yfff14 <sub>HEX</sub>  | Yfff16 <sub>HEX</sub>  | Yfff18 <sub>HEX</sub>  | Yfff1a <sub>HEX</sub>             | -                      | -                      |
| 2             | Yfff20 <sub>HEX</sub>        | Yfff22 <sub>HEX</sub>  | Yfff24 <sub>HEX</sub>  | Yfff26 <sub>HEX</sub>  | Yfff28 <sub>HEX</sub>  | Yfff2a <sub>HEX</sub>             | -                      | -                      |
| 3             | Yfff30 <sub>HEX</sub>        | Yfff32 <sub>HEX</sub>  | Yfff34 <sub>HEX</sub>  | Yfff36 <sub>HEX</sub>  | Yfff38 <sub>HEX</sub>  | Yfff3a <sub>HEX</sub>             | -                      | -                      |
| 4             | Yfff40 <sub>HEX</sub>        | Yfff42 <sub>HEX</sub>  | Yfff44 <sub>HEX</sub>  | Yfff46 <sub>HEX</sub>  | Yfff48 <sub>HEX</sub>  | Yfff4a <sub>HEX</sub>             | -                      | -                      |
| 5             | Yfff50 <sub>HEX</sub>        | Yfff52 <sub>HEX</sub>  | Yfff54 <sub>HEX</sub>  | Yfff56 <sub>HEX</sub>  | Yfff58 <sub>HEX</sub>  | Yfff5a <sub>HEX</sub>             | -                      | -                      |
| 6             | Yfff60 <sub>HEX</sub>        | Yfff62 <sub>HEX</sub>  | Yfff64 <sub>HEX</sub>  | Yfff66 <sub>HEX</sub>  | Yfff68 <sub>HEX</sub>  | Yfff6a <sub>HEX</sub>             | -                      | -                      |
| 7             | Yfff70 <sub>HEX</sub>        | Yfff72 <sub>HEX</sub>  | Yfff74 <sub>HEX</sub>  | Yfff76 <sub>HEX</sub>  | Yfff78 <sub>HEX</sub>  | Yfff7a <sub>HEX</sub>             | -                      | -                      |
| 8             | Yfff80 <sub>HEX</sub>        | Yfff82 <sub>HEX</sub>  | Yfff84 <sub>HEX</sub>  | Yfff86 <sub>HEX</sub>  | Yfff88 <sub>HEX</sub>  | Yfff8a <sub>HEX</sub>             | -                      | -                      |
| 9             | Yfff90 <sub>HEX</sub>        | Yfff92 <sub>HEX</sub>  | Yfff94 <sub>HEX</sub>  | Yfff96 <sub>HEX</sub>  | Yfff98 <sub>HEX</sub>  | Yfff9a <sub>HEX</sub>             | -                      | -                      |
| 10            | Yfffa0 <sub>HEX</sub>        | Yfffa2 <sub>HEX</sub>  | Yfffa4 <sub>HEX</sub>  | Yfffa6 <sub>HEX</sub>  | Yfffa8 <sub>HEX</sub>  | Yfffaa <sub>HEX</sub>             | -                      | -                      |
| 11            | Yfffb0 <sub>HEX</sub>        | Yfffb2 <sub>HEX</sub>  | Yfffb4 <sub>HEX</sub>  | Yfffb6 <sub>HEX</sub>  | Yfffb8 <sub>HEX</sub>  | Yfffb <sub>a</sub> <sub>HEX</sub> | -                      | -                      |
| 12            | Yfffc0 <sub>HEX</sub>        | Yfffc2 <sub>HEX</sub>  | Yfffc4 <sub>HEX</sub>  | Yfffc6 <sub>HEX</sub>  | Yfffc8 <sub>HEX</sub>  | Yfffc <sub>a</sub> <sub>HEX</sub> | -                      | -                      |
| 13            | Yffffd0 <sub>HEX</sub>       | Yffffd2 <sub>HEX</sub> | Yffffd4 <sub>HEX</sub> | Yffffd6 <sub>HEX</sub> | Yffffd8 <sub>HEX</sub> | Yffffda <sub>HEX</sub>            | -                      | -                      |
| 14            | Yffffe0 <sub>HEX</sub>       | Yffffe2 <sub>HEX</sub> | Yffffe4 <sub>HEX</sub> | Yffffe6 <sub>HEX</sub> | Yffffe8 <sub>HEX</sub> | Yffffea <sub>HEX</sub>            | Yffffec <sub>HEX</sub> | Yffffee <sub>HEX</sub> |
| 15            | Yfffff0 <sub>HEX</sub>       | Yfffff2 <sub>HEX</sub> | Yfffff4 <sub>HEX</sub> | Yfffff6 <sub>HEX</sub> | Yfffff8 <sub>HEX</sub> | Yfffffa <sub>HEX</sub>            | Yfffffc <sub>HEX</sub> | Yfffffe <sub>HEX</sub> |

Slika 7. 6: Parametrski RAM TPU-ja

## 7.2 Primer uporabe PWM funkcije

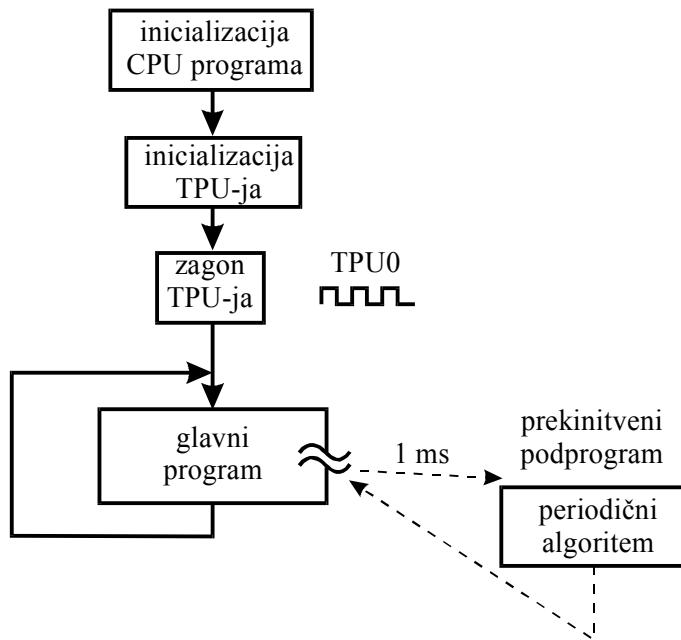
### 7.2.1 Princip delovanja programa

Opis konfiguriranja TPU-ja, izbire vseh funkcij in definiranja potrebnih parametrov zahteva preveč prostora, zato bomo tukaj opisali karakteristični zgled uporabe funkcije PWM (za ostale funkcije glej [26]).

Naloga, ki jo želimo realizirati, je splošno konfiguriranje TPU-ja ter definiranje in zagon funkcije PWM na kanalu TPU0, ki bo generirala neskončni vlak pulzov s periodom 1 ms. Po zagonu delujeta CPU in TPU neodvisno. Po preteku vsake periode naj TPU prekine delovanje CPU, ki bo takrat izvedel prekinitveni podprogram (njegovo trajanje je krajše od 1 ms!). Po vrnitvi iz podprograma naj CPU nadaljuje s svojim dotedanjim delom do naslednje prekinitve.

Očitno bomo funkcijo PWM v tem kontekstu uporabili za generiranje periodične prekinitve v trajanju 1 ms. Na ta način lahko zagotovimo periodičnost izvajanja prekinitvenega podprograma s časom vzorčenja  $T_v = 1$  ms (glej tudi pogl. 10). Slika 7. 7 kaže posamezne korake, ki jih je treba izvršiti v CPU programu:

1. Inicializacija splošnih spremenljivk, ki jih uporabljam v programu
2. Inicializacija TPU registrov, definiranje PWM funkcije na kanalu TPU0 ter njena parametrizacija; omogočanje TPU prekinitve.
3. Zagon TCR1 in PWM funkcije s periodom 1 ms.
4. Izvajanje glavnega CPU programa (manj zahtevne funkcije, npr. komunikacija z uporabnikom). Čakanje na prekinitvah. Čas izvajanja podprograma mora biti krajši od 1 ms.
5. Prekinitveni podprogram (v bistvu "glavni" program), ki se izvaja periodično s  $T_V = 1$  ms.



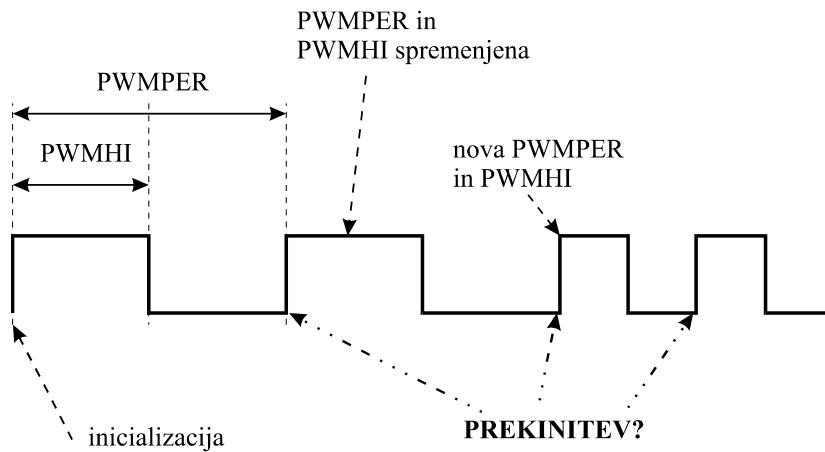
**Slika 7. 7: Diagram poteka generiranja periodičnega vzorčenja s pomočjo PWM**

### 7.2.2 Funkcija PWM s periodo 1 ms

PWM funkcija generira vlak pulzov (0 V, 5 V) na izhodu TPU kanala, (načeloma) spremenljive periode. Osnovna parametra funkcije sta na sliki 7. 8 označena s simboloma PWMPER in PWMHI in določata trajanje periode ter trajanje stanja 1 (5 V). Oba parametra podajamo v obliki mnogokratnikov periode osnovnega takta (TCR1 ali TCR2), med obratovanjem pa ju lahko spremojemo<sup>3</sup>. Pri vsakem prehodu 0 → 1 lahko kanal prekine izvajanje programa v CPU.

V našem primeru bosta PWMPER in PWMHI konstantna, ker pa bo namen te funkcije le generiranje periodične prekinitve, vrednost PWMHI nima nobenega pomena (vsekakor pa mora veljati PWMPER > PWMHI).

<sup>3</sup> Od tod tudi naziv PWM - pulzno-širinska modulacija. Možna aplikacija je generiranje spremenjajoče se napetosti ob konstantnem PWMPER in spremenjajočem se PWMHI. Na binarnem izhodu bomo po filtriraju z nizkopasovnim filtrom dobili "zvezno" napetost.

*Slika 7. 8: Definiranje dolžine pulzov v funkciji PWM*

#### 7.2.2.1 Konfiguriranje in parametrirvanje funkcije PWM

Po izbiri časovne funkcije v registru CFSR (za PWM je to koda 9<sub>HEX</sub>, glej pogl. 7.1.1.5) moramo izbrati še nekatere druge, za to funkcijo značilne konfiguracijske nastavitev (v HSQR in HSRR registrih, pogl. 7.1.1.6 in 7.1.1.7) in parametre (v parametrskem RAM-u za posamezne kanale).

V primeru PWM register HSQR nima nobenega pomena, bita v HSRR0 ali HSRR1 za posamezne kanale pa ponujata naslednje opcije (glej tudi sliko 7. 5):

- 00 brez servisiranja ali servisiranje končano (npr. inicializacija)
- 01 takojšnje ažuriranje PWM
- 10 inicializacija.

Parametrski RAM posameznega kanala za PWM funkcijo kaže slika 7. 9.

| naslovi (HEX)       | mnemonik | 15 | 9 | 8               | 0 |
|---------------------|----------|----|---|-----------------|---|
| YfffW0 <sup>4</sup> | TPUW_0   |    |   | CHANNEL_CONTROL | 0 |
| YfffW2              | TPUW_2   |    |   | OLDRIS          |   |
| YfffW4              | TPUW_4   |    |   | PWMHI           |   |
| YfffW6              | TPUW_6   |    |   | PWMPER          |   |
| YfffW8              | TPUW_8   |    |   | PWMRIS          |   |

*Slika 7. 9: Lokacije v parametrskem RAM posameznega kanala za funkcijo PWM*

<sup>4</sup> W je oznaka kanala, na katerega se nanaša funkcija, npr. prvi parameter kanala 5 se nahaja na TPU RAM lokaciji Yfff50<sub>HEX</sub>, drugi na Yfff52<sub>HEX</sub> itd.

Parametra PWMHI in PWMPER že poznamo. Oba sta 16-bitna (največja vrednost 65535) mnogokratnika TCR1 ali TCR2. OLDRIS je čas, pri katerem je prišlo do zadnjega prehoda 0 → 1, PWMRIS pa trenutek nastopa naslednjega prehoda (OLDRIS + PWMPER), ki se izračuna na začetku vsakega pulza.

Parameter CHANNEL\_CONTROL je 9-bitno polje (slika 7. 10), v katerem določimo, katero izmed dveh ur (TCR1 ali TCR2) bomo upoštevali pri podajanju PWMPER in PWMHI, ter začetno stanje izhoda po inicializaciji (polje PSC).

| TBS |   |   |   | PAC | PSC | Dejavnost |                  |                                 |                  |
|-----|---|---|---|-----|-----|-----------|------------------|---------------------------------|------------------|
| 8   | 7 | 6 | 5 | 4   | 3   | 2         | 1    0           | Vhod                            | Izhod            |
|     |   |   |   |     | 0   | 0         | -                | Postavi nožico kot določa PAC   |                  |
|     |   |   |   |     | 0   | 1         | -                | Postavi nožico na "1"           |                  |
|     |   |   |   |     | 1   | 0         | -                | Postavi nožico na "0"           |                  |
|     |   |   |   |     | 1   | 1         | -                | Ne spreminja stanja             |                  |
|     |   |   |   | 1   | x   | x         |                  | Ne spreminja PAC                | Ne spreminja PAC |
| 0   | 1 | x | x |     |     |           | -                | <b>Izhodni kanal</b>            |                  |
| 0   | 1 | 0 | 0 |     |     |           | -                | Zajemanje TCR1, primerjava TCR1 |                  |
| 0   | 1 | 1 | 1 |     |     |           | -                | Zajemanje TCR2, primerjava TCR2 |                  |
| 1   | 1 | x | x |     |     |           | Ne spreminja TBS | Ne spreminja TBS                |                  |

x je poljubna vrednost

*Slika 7. 10: Pomen bitov v parametru CHANNEL\_CONTROL funkcije PWM*

### 7.2.2.2 Incializacija TPU-ja

O programu za inicializacijo TPU parametrov, napisanem v C jeziku, bo več govora v nadaljevanju (glej tudi sliko 7. 11). Na tem mestu si oglejmo nastavitev posameznih registrov. Pri delu bomo uporabljali njihove standardne simbole oz. kratice, ki jim moramo na začetku programa dodeliti ustrezne pomnilniške lokacije (glej tudi slike 7. 2 in 7. 6).

#### TPUMCR

TPUMCR smo postavili v stanje  $6041_{HEX} = 0110\ 0000\ 0100\ 0001_{BIN}$ . Na ta način smo definirali periodo ure TCR1, ki bo približno enaka  $2\ \mu s$  (pogl. 7.1.1.1)<sup>5</sup>.

#### CFSR3

Vsebina registra CFSR3 je  $9_{HEX} = 0000\ 0000\ 0000\ 1001_{BIN}$ , s tem smo pa definirali funkcijo PWM na kanalu TPU0 (vsi ostali kanali so neaktivni; glej tudi sliko 7. 5).

#### CPR1

<sup>5</sup> V praksi se ta podatek pogostokrat nekoliko razlikuje od deklariranega, zato je treba v primerih ko želimo zelo natančno določiti periodo, opraviti ustrezne meritve in korigiranje nastavitev.

$(CPR1) = 3_{HEX} = 0000\ 0000\ 0000\ 0011_{BIN}$ . Kanal TPU0 ima najvišjo prioriteto.

## CIER

$(CIER) = 1_{HEX} = 0000\ 0000\ 0000\ 0001_{BIN}$ . S tem smo kanalu TPU0 omogočili generiranje prekinitve (za PWM funkcijo je to ob vsakokratnem prehodu z 1 na 0).

## TICR

$(TICR) = 0640_{HEX} = 0000\ 0110\ 0100\ 0000_{BIN}$ . V podoglavlju 7.1.1.2 o registru za konfiguracijo prekinitvev smo omenili njegovo dvojno vlogo: določanje nivoja zahteve po prekinitvi in baznega prekinitvenega vektorja. Navedena vsebina registra določa drugo po vrsti prekinitveno prioriteto (biti 8,9,10 =  $110_{BIN} = 6$ ; najvišja je 7). Bazni prekinitveni vektor je v tem primeru 4, kar pomeni, da je zaporedna številka prekinitveni vektor za TPU0  $40_{HEX}$ , za TPU1  $41_{HEX}$ , za TPU15 pa  $4f_{HEX}$ . Konkretne lokacije vektorjev za posamezne kanale v RAM pomnilniku izračunamo z znano formulo (glej tudi proceduro `set_handler` na sliki 7. 11 ter poglavje o TICR registru):

$$\begin{aligned} (VBR) + 4 \cdot 40_{HEX} &= (VBR) + 4 \cdot 64_{DEC} && \text{za TPU0,} \\ (VBR) + 4 \cdot 41_{HEX} &= (VBR) + 4 \cdot 65_{DEC} && \text{za TPU1,} \\ (VBR) + 4 \cdot 4f_{HEX} &= (VBR) + 4 \cdot 79_{DEC} && \text{za TPU15 itd.} \end{aligned}$$

Tako definirani vektorji TPU-ja zasedejo naslednje lokacije v vektorski mapi:

| Št. vektorja | Premik (offset) |       | Opis         |
|--------------|-----------------|-------|--------------|
|              | dec             | hex   |              |
| 0            | 0               | 000   | Reset: SP    |
| 1            | 4               | 004   | Reset: PC    |
| .....        | .....           | ..... | .....        |
| 64           | 256             | 100   | TPU kanal 0  |
| 65           | 260             | 104   | TPU kanal 1  |
| 66           | 264             | 108   | TPU kanal 2  |
| 67           | 268             | 10C   | TPU kanal 3  |
| 68           | 272             | 110   | TPU kanal 4  |
| 69           | 276             | 114   | TPU kanal 5  |
| 70           | 280             | 118   | TPU kanal 6  |
| 71           | 284             | 11C   | TPU kanal 7  |
| 72           | 288             | 120   | TPU kanal 8  |
| 73           | 292             | 124   | TPU kanal 9  |
| 74           | 296             | 128   | TPU kanal 10 |
| 75           | 300             | 12C   | TPU kanal 11 |
| 76           | 304             | 130   | TPU kanal 12 |
| 77           | 308             | 134   | TPU kanal 13 |
| 78           | 312             | 138   | TPU kanal 14 |
| 79           | 316             | 13C   | TPU kanal 15 |

## TPU0\_0

Preden končamo z inicializacijo, moramo določiti še parametre kanala (tukaj TPU0) za izbrano funkcijo (glej slike 7. 9 in 7. 10). V konkretnem primeru je  $TPU0\_0 = 0091_{HEX} = 0000\ 0000\ 1001\ 0001_{BIN}$ , torej smo izbrali TCR1 ter začetno stanje kanala TPU0 1.

## TPU0\_4 in TPU\_6

TPU0\_4 in TPU\_6, torej PWMHI in PWMPER postavimo v stanje  $524_{DEC}$  in  $250_{DEC}$ . To sta mnogokratnika periode izbranega TCR1 ( $\approx 2 \mu s$ ; glej register TPUMCR). Trajanje PWM periode bo potem takem 1 ms, signala 1 pa približno 0,5 ms (vrednost 524 in ne 500 je izbrana po natančni meritvi trajanja periode TCR1).

## HSRR1

TPU kanal 0 je konfiguriran in parametriran. Sedaj preostaja le inicializacija kanala, ki jo sproži CPU s postavitvijo HSRR1 v stanje  $2 = 0000\ 0000\ 0000\ 0010_{BIN}$  (glej sliko 7. 5). Postopek zahteva določeno število urinih ciklov, edino zagotovilo o končani proceduri pa je samodejno postavljanje teh bitov v stanje  $00_{BIN}$ . Zaradi tega je v program vstavljena pogojna zanka, ki onemogoča nadaljevanje programa preden je pogoj ( $HSRR1 = 0$ ) izpolnjen.

### 7.2.2.3 Glavni in prekinitveni program

Po končani inicializacijski proceduri oz. zagonu TPU-ja, nadaljuje CPU z obdelovanjem “glavnega programa” do nastopa časovne prekinitve. V našem zgledu je glavni program le neskončna zanka (v proceduri `main`). Pri prekinitvi pride do skoka v prekinitveni podprogram (`handler`). Načeloma je v njem dejanski regulacijski program, v tem primeru pa le resetiramo prekinitveni statusni bit kanala TPU0 in s tem omogočimo ponovno prekinitvev. Ponovimo še enkrat, da mora celotno trajanje tega podprograma biti krajše od časa med zaporednima prekinitvama, drugače lahko pride do nenadzorovanega dogajanja (glej tudi zgled v pogl. 10).

### 7.2.3 C program za generiranje periodične prekinitve

Delovanje programa smo v grobem že opisali. Čeprav so funkcije relativno transparentne, omenimo tukaj le nekatere konvencije C jezika, ki jih v programu uporabljamo:

- Prefiks `0x` določa šestnajstiško notacijo števila.
- Komentar se nahaja znotraj oznak `/*` in `*/`.
- *Kazalci* (angl. pointers; simbol `*`) so zelo eleganten način naslavljanja spominskih lokacij. Velja: če je `TPUMCR` naslov neke lokacije (tukaj `0xfffffff0`) je `*TPUMCR` vsebina te lokacije. Večina deklaracij `#define` v programu dodeljuje simbole naslovom spominskih lokacij (npr. `TPUMCR`). Kazalci na te naslove (`*TPUMCR`) imajo dostop le do formata, ki je vsebovan v deklaraciji (za `TPUMCR` je to t.i. `unsigned short int` oz. 16-bitni celoštevilski format brez predznaka).

- V C-ju pogostokrat uporabljana skrajšana oblika ukaza, npr. `*CISR &= 0xffffe` (velja tudi za ostale podobne logične in aritmetične operacije), je enaka bolj znani sintaksi `*CISR = *CISR & 0xffffe`.
- Prvi del programa je le definiranje simbolov, najprej simboličnih lokacij, potem pa konstant.
- Temu sledijo deklaracije podprogramov: `init` (inicIALIZacija TPU-ja), `set_handler` (definiranje prekinitvenega vektorja za TPU0), `handler` (prekinitveni program; angl. "handler" - upravljač - je običajen naziv za prekinitvene podprograme) in `main` (glavni program). V tem delu so eventualni vhodni parametri (npr. `set_handler(handler, vecno)`) podani le simbolično. Šele pri klicu posamezne rutine jim dodelimo dejanske vrednosti.
- Če smo v C-ju definirali proceduro npr. `handler()`, je spremenljivka `handler` (brez oklepajev!) naslov, na katerem se ta procedura nahaja.
- `set_handler` je procedura za postavljanje naslova prekinitvenega programa (`handler`) na prekinitveni vektor z zaporedno številko  $40_{\text{HEX}} = 64_{\text{DEC}}$ . Oba podatka sta vhodna parametra pri klicu te procedure: `set_handler(handler, b4)`.
- Maskiranje: pogostokrat želimo določene bite v registru ali spominski lokaciji postaviti v stanje 1 ali 0, ne glede na stanje ostalih bitov (npr. glej CISR register v programu). To dosežemo s t.i. "maskiranjem": če gre za postavitev bitov v stanje 0, naredimo to z operacijo "logični IN" (angl. AND, simbol `&`), v stanje 1 pa z operacijo "logični ALI" (angl. OR, simbol `|`). V primeru iz programa `(*CISR &= 0xffffe` oz. zaradi boljšega pregleda:  $1111\ 1111\ 1111\ 1110_{\text{BIN}}$ ) se v stanje 0 postavi le bit 0, vsem ostalim pa se vsebina ne spremeni. Če bi hoteli npr. le bit 3 gotovo postaviti v stanje 1, bi to lahko storili z ukazom `*CISR |= 0x0008` ( $0000\ 0000\ 0000\ 1000_{\text{BIN}}$ ). **PAZI:** selektivnega postavljanja bitov z običajnim dodeljevanjem (simbol `=`) ne moremo doseči, ker s tem nujno postavimo vse bite v določeno stanje!
- Ukaz ali ukazi (v obehajih {}) v zanki `while(x)` se izvajajo toliko časa, dokler je izpolnjen pogoj `x`. Ukaz za zanko ne zaključujemo s podpičjem ";" kot navadne ukaze. C pozna tudi "prazne" ukaze, ki se ne izvršujejo, kot vsak pravi ukaz pa se morajo končati s podpičjem. Npr. ;;; so trije takšni ukazi. V našem primeru imamo opravka z zanko "`while(1) ;`", torej v neskončni zanki ne izvaja nič, razen ponovnega preverjanja izpolnjenosti pogoja.

```
/* vajevanje C knjiznic */

#include <stdio.h>
#include <stdlib.h>

#define TPUMCR (unsigned short int*) 0xfffffe00 /* module config. reg. */
#define TICR (unsigned short int*) 0xfffffe08 /* interrupt conf. reg. */
#define CIER (unsigned short int*) 0xfffffe0a /* interrupt enable reg. */
#define CISR (unsigned short int*) 0xfffffe20 /* interrupt status reg. */
#define CFSR3(unsigned short int*) 0xfffffe12 /* chann. fn. sel. reg. */
#define HSRR1(volatile unsigned short int*) 0xfffffe1a /* host service
req.reg. */
#define CPR1 (unsigned short int*) 0xfffffe1e /* chann. prior. reg. */
#define TPU0_0 (unsigned short int*) 0xfffffe00 /* channel control */
#define TPU0_4 (unsigned short int*) 0xfffffe04 /* naslov PWMHI */
#define TPU0_6 (unsigned short int*) 0xfffffe06 /* naslov PWMPER */

#define PWMPER 524 /* 524 x 1.9 us = 1 ms */
```

```
#define PWMHI 250

/*********************************************
/*          INICIALIZACIJA TPU-ja           */
/********************************************

void init()
{
    *TPUMCR=0x6041;
    *CFSR3=0x0009; /* izbrana funkcija */
    *CPR1=0x0003; /* nastavitev prioritet */
    *CIER=0x0001; /* omogoca prekinitev na TPO */
    *TICR=0x640; /*nastavitev nonmaskable prekintive, lokacija
                    prekinitvenega vektorja*/
    *TPU0_0=0x0091; /* nastavitev channel control */
    *TPU0_6=PWMPPER; /* trajanje periode PWMPPER */
    *TPU0_4=PWMHI; /* trajanje PWMHI */

    *HSRR1=0x0002; /* inicializacija TPU; izvaja PWM */

    while (*HSRR1 != 0x0000); /* cakamo do konca inicializacije
                                takrat je *HSRR1=0x0000 */
    *CISR &= 0xffff;

}/* konec init */

/*********************************************
/*      PODPROGRAM ZA ZAPIS ZA ČETNEGA NASLOVA PREKINITVENEGA      */
/*      PODPROGRAMA (SPREM handler JE POINTER NA PODPROGRAM        */
/*      handler()) NA LOKACIJO Z NASLOVOM (VBR)+4*vecno          */
/********************************************

void set_handler(handler,vecno) /*handler je definirani prek.
podprogram,                                     vecno=64*/
{
    void(*handler);
    int vecno;

    *((void**) (0) (4*vecno))=handler;
}

/* konec set_handler */

/*********************************************
/*          PREKINITVENI PODPROGRAM           */
/********************************************

_SWI void handler()
{

/* namesto tega komentarja pride "glavni" regulacijski program */

    *CISR &= 0xffff; /* omogocanje ponovne prekinitve */

}/* konec handler */

/*********************************************
/*          GLAVNI PROGRAM                 */
/********************************************

main()
```

```
{  
    set_handler(handler,b4); /* definicija vektorja za int. handler */  
    init();                  /* klic funkcije za inicializacijo TPU-ja */  
    while(1);                /* neskončna zanka; pogoj vedno izpolnjen */  
/*/  
}/* konec main*/
```

*Slika 7. 11: Program v C-ju za generiranje periodične prekinitve s pomočjo TPU-ja*

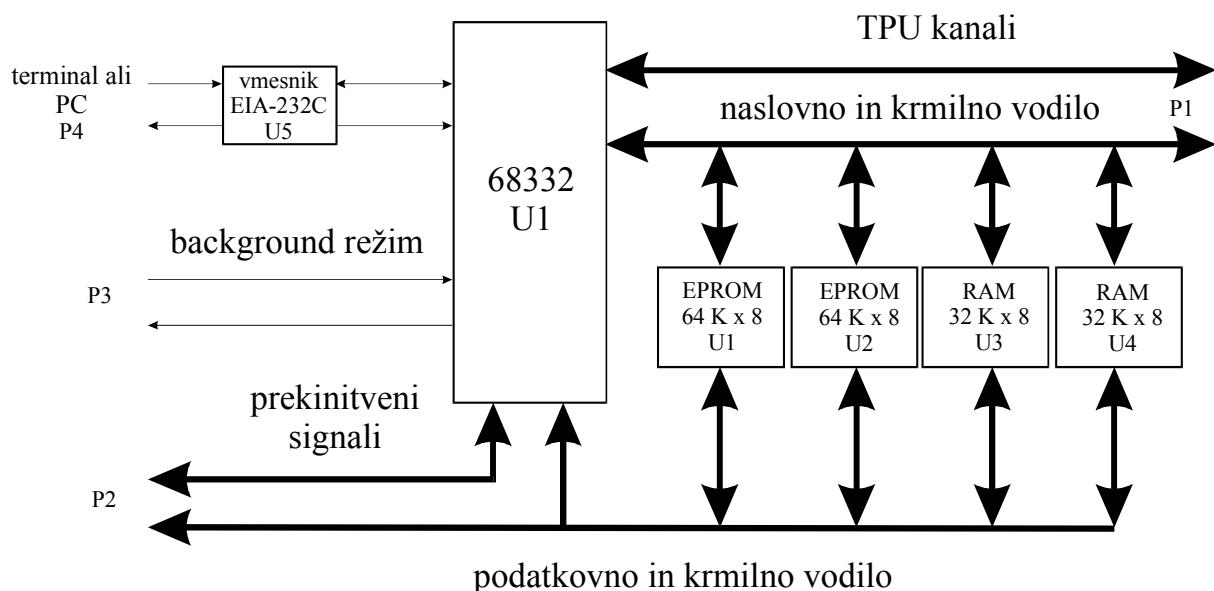
## 8. MC68332 V BCC IZVEDBI

V tem poglavju bomo pokazali praktično izvedbo arhitekture mikrokrmlniškega sistema z nekaterimi osnovnimi podsestavi. Takšne elementarne sisteme, ki imajo splošno namembnost ponujajo skorajda vsi proizvajalci procesorjev. V primeru MC68332 govorimo o BCC-ju (angl. Business Card Computer - računalnik velikosti vizitke), ki skupaj s PFB (angl. Platform Board) sestavlja razvojni modul (angl. Evaluation Module). Podobni sistemi so na voljo tudi pri ostalih procesorjih (npr. EVM DSP-ja TMS320F240, pogl. 11). Taki aparaturni opremi je prilagojena tudi programska oprema, ki omogoča najosnovnejše delo s sistemom.

### 8.1 BCC

BCC je sistem, ki na tiskanem vezju dimenzijs 90 mm x 60 mm poleg mikrokrmlnika MC68332 vsebuje še nekatera dodatna vezja in elemente (slika 8.1):

- 2 x 64 K byte EPROM pomnilnik z debug monitor programom (glej poglavje 8.2),
- 2 x 32 K byte RAM pomnilnik,
- vmesnik za serijsko komunikacijo EIA-232,
- dva konektorja po 64 nožic.

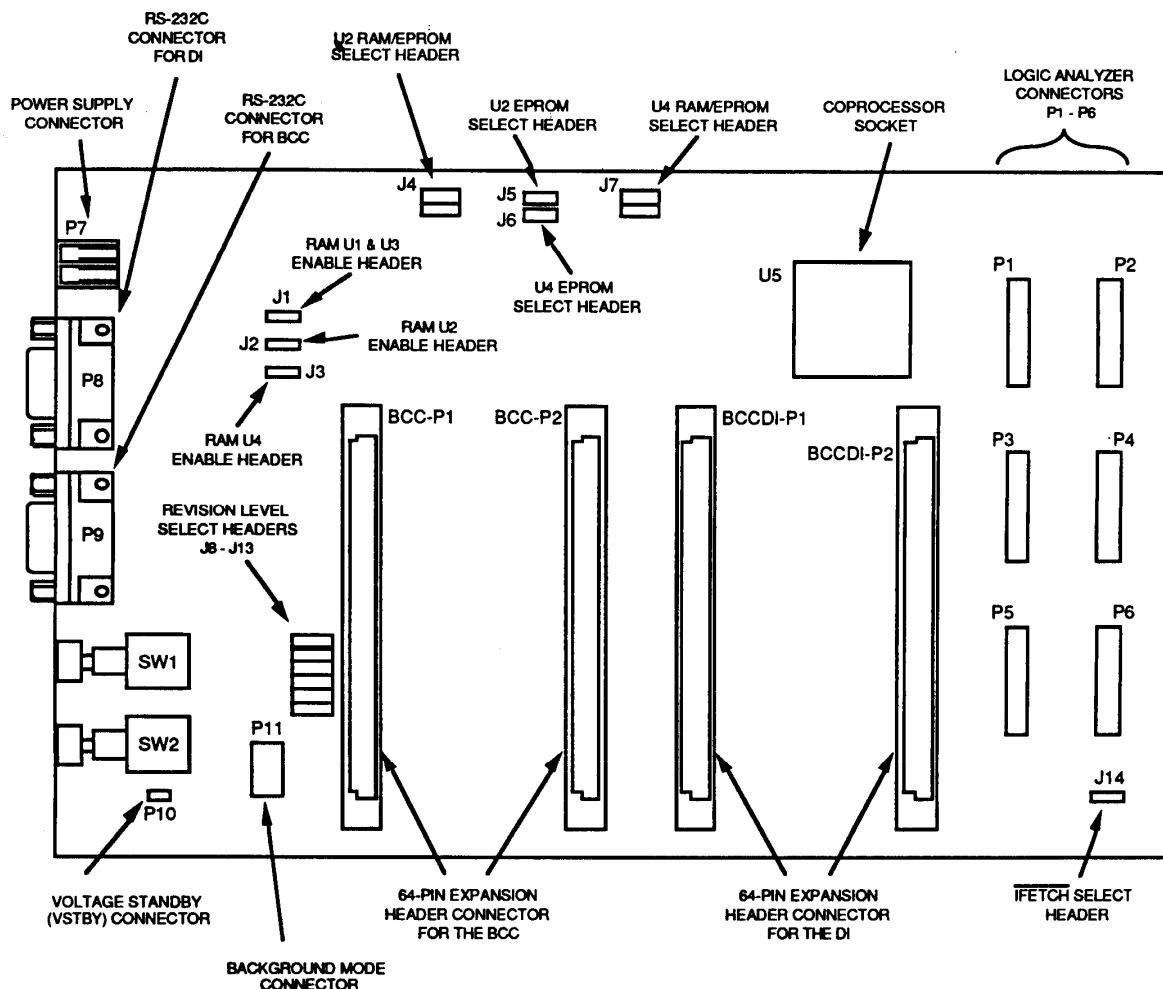


*Slika 8.1: Blokovna shema BCC za MC68332*

BCC lahko vgradimo v večji mikroračunalniški sistem, kar omogočata konektorja na robovih ploščice. Prek njiju so dostopne vse nožice mikrokrmlnika. Nadgraditev BCC je enostavna: na tiskanem vezju, ki vsebuje poljubne dodatne elemente, je treba predvideti konektorja, ki sta združljiva s konektorjem na BCC-ju. Iz tega sledi, da lahko BCC uporabimo za vrsto različnih nalog, potrebno je le projektirati nosilno ploščo z dodatnim okoljem (dodatni pomnilniki, A/D in D/A pretvorniki, prilagodilna vezja itd.).

Tudi Motorola ponuja tiskano vezje, ki je aparatura podpora BCC-ju. Imenujejo ga Platform Board - PFB. Slika 8.2 kaže skico PFB-ja. Osnovni elementi so:

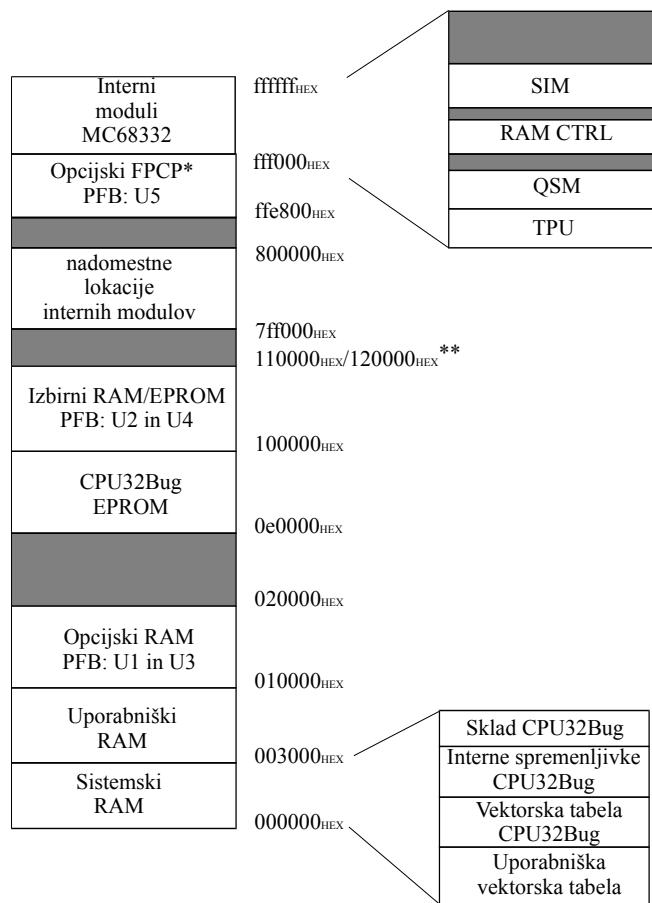
- priključne sponke za napajalno napetost 5 V z varovalko in stabilizacijskim vezjem,
- dva konektorja z devetimi nožicami za priključitev na serijski port (EIA232) osebnega računalnika (konektor TERMINAL je povezan s SCI-jem BCC-ja),
- podnožja za dodatne EPROM in RAM pomnilnike in matematični koprocesor (angl. floating point processor - FPP),
- konektorja za priključitev BCC-ja,
- konektorja za priključitev M68300DI (angl. Development Interface),
- premostitve (angl. jumper) za izbiro med različnimi arhitektturnimi inačicami (npr. izbira tipa pomnilnika - RAM ali EPROM - na določenem podnožju),
- številni posebni konektorji z dvajsetimi nožicami, ki omogočajo dostop do vseh signalov BCC-ja za priključitev logičnega analizatorja (signali so razvrščeni v različne skupine, npr. naslovno vodilo, podatkovno vodilo, TPU kanali itd.).
- Tipki za resetiranje (SW1) in prekinitve izvajanja programa (abort - SW2). Prva tipka je neposredno povezana s nožico RESET prekinitve, druga pa z IRQ7.



Slika 8.2: Skica PFB-ja

### 8.1.1 Pomnilniška mapa MC68332 na BCC-ju

Pomnilniki na BCC-ju ter morebitni dodatni pomnilniki na PFB-ju in matematični koprocesor se nahajajo na vnaprej določenih naslovih. S tem je del področja naslavljanja mikrokrmilnika MC68332 že rezerviran (celotno naslovljivo področje z naslovnimi nožicami A23 - A0 je  $2^{24} = 16\,777\,216$  bytov). Prikaz zasedenosti naslovljivega področja imenujemo *pomnilniška mapa* (angl. memory map, slika 8.3). Dodatne naslovljive elemente v končni aplikaciji (pomnilnike, A/D in D/A pretvornike...) lahko postavimo le na prosta področja. Pomnilniška mapa velja za najnovejšo inačico BCC-ja in se nekoliko razlikuje od starejših verzij.



\*FPCP: matematični koprocesor (angl. Floating Point Coprocessor)

\*\*odvisno od tipa pomnilnika

*Slika 8.3: Pomnilniška mapa BCC*

## 8.2 Program debug monitor

Večina mikrokrmilnikov vsebuje številne podsestave, ki zadostujejo vsaj za osnovno delovanje (CPU, enota za komunikacijo z nadrejenim računalnikom, pomnilnik, vhodno/izhodne enote). Kljub temu samega integriranega vezja po priključitvi napajanja ne

moremo takoj uporabljati. Mikroračunalnik (ali katerikoli drug mikroprocesorski element) se obnaša kot "tabula rasa"; nihče mu ni povedal kako naj uporablja svoje zmogljivosti. Določitev osnovnih začetnih parametrov je glavna naloga *inicializacijskega postopka*. To je del podprograma, ki se izvaja avtomatično ob vklopu ali po *resetiranju* procesorja. Aktiviranje RESET vhoda generira prekinitvev najvišje prioritete, ki je ni možno maskirati. Ob tem se samodejno začne izvajati prekinutveni program, katerega naslov kaže vektor 1 (pomnilniška lokacija 00 0004<sub>HEX</sub>), kazalec sklada (SP) za ta podprogram pa je zapisan v vektorju 0 (pomnilniška lokacija 00 0000<sub>HEX</sub>, glej tudi poglavje 7.2.2.2).

Med tipične naloge, ki jih je treba izvršiti med inicializacijo, sodi določanje:

- parametrov serijskega podsestava za asinhronko komunikacijo z nadrejenim računalnikom (PC) ali terminalom (hitrost prenosa, število bitov komunikacijskega paketa, tip paritete, število stop bitov itd.),
- začetnega stanja vseh registrov (npr. stanje SP-ja in PC-ja ob zaključku izvajanja inicializacije) in nožic,
- sistemskih prekinutvenih podprogramov (npr. procedura, ki se izvaja ob naslavljaju neobstoječe spominske lokacije) itd.

Vse inicializacijske funkcije morajo biti shranjene na mediju, ki ohrani svojo vsebino tudi po izklopu napajanja. V ta namen običajno uporabljamo pomnilnike tipa ROM, največkrat EPROM, ki omogoča reprogramiranje inicializacijskega podprograma. Načeloma mora uporabnik pri projektiranju mikroprocesorskega vezja postaviti pomnilnik z inicializacijskim programom na lokacijo, ki ustreza RESET vektorju, tako da se program samodejno požene ob vklopu ali resetiranju.

Problem izbire EPROM-a pri MC68332 ob resetiranju je rešen s pomočjo posebnega signala CSBOOT, ki sodi v skupino CS nožic (glej poglavje 5.5). Na njo je vezan vhod G EPROM-a z inicializacijskim programom. Po RESET signalu se izmed vseh nožic za izbiro čipa aktivira samo CSBOOT (stanje "0"). Na ta način omogoči EPROM mikrokrumilniku dostop do svojih prvih osmih bytov (RESET SP in PC). Vsebina prekinutvenega vektorja 1 (nov PC) kaže na RESET prekinutveni program, ki se nahaja na istem EPROM-u.

ABORT tipka aktivira prekinitvev IRQ7, ki le prekine izvajanje uporabniškega programa in vrne nadzor debug monitorskemu programu. Pri tem pa ABORT ne reinicializira sistema, kot to storii funkcija RESET.

### 8.2.1 Naloge debug monitor programa

Incializacija je le ena od nalog *debug monitor* programa CPU32Bug v EPROM-u. Program je dobil ime po dveh svojih osnovnih funkcijah:

- "**debugging**" (slov. očiščevanje, razhroščevanje) pomeni preizkušanje, odkrivanje in odpravljanje napak v programu,
- **monitor** (slov. monitor, nadzorni program) je del programa, ki vsebuje spremenljivke, procedure za dostop in inicializacijsko kodo.

Program vsebuje ukaze za:

- prikaz in spremiščanje vsebine pomnilnikov,

- določanje prekinitvev točk,
- asemlirjanje in deasembliranje,
- samodejno preverjanje funkcionalnosti sistema ob vklopu,
- interaktivni razhroščevalnik (angl. debugger),
- uporabniški vmesnik, ki sprejema ukaze s terminala oz. osebnega računalnika.

S tem je omogočeno delo z BCC-jem v t.i. *on-line režimu*, kjer lahko v RAM pomnilnik na BCC neposredno vpisujemo ali beremo ukaze in podatke, prenašamo dele vsebine pomnilnika na druge lokacije, zaženemo program, preverjamo stanje določenih registrov, vnašamo določene *prekinitvene točke* (angl. breakpoints) itd.

#### 8.2.1.1 Dekodiranje S19 formata

V poglavju 9 je opisan celoten postopek programiranja uporabniškega programa v “off-line” režimu, to je na osebnem računalniku, brez neposredne povezave z mikroprocesorskim sistemom. Postopek se začne z izvorno kodo (v C jeziku ali zbirniku) in konča z ASCII datoteko, ki je primerna za prenos v RAM pomnilnik mikroprocesorskega sistema (angl. downloading). Format te datoteke za Motoroline procesorje imenujemo “S19” ali “HEX”. Datoteko ne sestavlja le bodoča vsebina RAM-a v ASCII obliku (kodirani ukazi in podatki), temveč tudi pomožni podatki (naslov programa, začetne lokacije memorijskih paketov, število zlogov v vrstici, nadzorna vsota - checksum itd.). Seveda mora tudi sprejemnik S19 datoteke (mikroprocesorski sistem, v katerega nalagamo datoteko, npr. BCC) ločiti med delom, ki ga je treba postaviti v RAM in “balastom”, kar je ena od nalog debug monitor programa v BCC-ju.

#### 8.2.1.2 Asemlirjanje in deasembliranje

“On-line” režim omogoča neposredno vpisovanje posameznih ukazov in podatkov na lokacije v RAM-u prek terminala. Vpisovanje podatkov je možno v desetiški ali dvojiški obliku, najpogosteje pa uporabljamo šestnajstiški način zapisa.

Vsebina RAM-a s CPU ukazi (spoznali smo jih v poglavju 4) se na prvi pogled sploh ne razlikuje od dela RAM-a, ki vsebuje podatke, saj so vsi zapisani v binarni kodi. Seveda gre pri ukazih za binarno strojno kodo, ki ustreza le določenemu tipu ukaza in izbranega načina naslavljanja operandov. Kodiranje ukazov s strani programera in vpisovanje v RAM je možno, vendar je to zelo zahtevno in zamudno delo. Zato raje uporabljamo vrstični<sup>1</sup> zbirnik ali *asembler* (angl. assembler), ki je del debug monitorja. Ta program omogoča prevajanje CPU ukazov iz mnemotehnične oblike (t.i. *zbirni jezik*; assembly language) v ustrezeno binarno kodo. Za čitanje ukazov rabimo *povratni zbirnik* ali *dezasembler* (angl. disassembler), ki binarno kodo ukaza prevede nazaj v mnemotehnično obliko.

---

<sup>1</sup> O vrstičnem (angl. line) zbirniku govorimo zaradi njegove lastnosti, da hkrati vnašamo oz. popravljamo le eno vrstico.

### 8.2.2 Debug monitorski ukazi

Funkcije debug monitor programa v EPROM-u BCC-ja izvajamo s pomočjo niza ukazov v mnemotehnični obliku. Teh ukazov ne gre v nobenem primeru mešati s CPU ukazi (Poglavlje 4). Debug monitorski ukazi le pomagajo pri manipulaciji s programom, ki je sestavljen iz CPU ukazov. Slednji se po zagonu (prek debug monitorskega ukaza **G0**) izvajajo v realnem času.

V nadaljevanju bomo ob mnemoniku vsakega ukaza v oklepajih zapisali tudi dovoljeno skrajšano obliko. Ukaze lahko pišemo z velikimi ali malimi črkami. Tukaj bomo prikazali le nekatere ukaze in njihove osnovne različice.

#### Ukaz za pomoč: HELP (HE)

Ukaz **HELP** izpiše vse debug monitor ukaze s kratkim opisom njihovega pomena:

|       |   |
|-------|---|
| BC    | Block Compare                             |
| BF    | Block Fill                                |
| BM    | Block Move                                |
| BR    | Breakpoint Insert                         |
| N0BR  | Breakpoint Delete                         |
| BS    | Block Search                              |
| BV    | Block Verify                              |
| DC    | Data Conversion and Expression Evaluation |
| DU    | Dump S-Records                            |
| GD    | Go Direct (no breakpoints)                |
| GN    | Go and Stop after Next Instruction        |
| G0    | Go to Target Code                         |
| G     | "Alias" for previous command              |
| GT    | Go and Insert Temporary Breakpoint        |
| HE    | Help Facility                             |
| L0    | Load S-Records                            |
| MA    | Macro Define/Display                      |
| N0MA  | Macro Delete                              |
| MAE   | Macro Edit                                |
| MAL   | Enable Macro Expansion Listing            |
| N0MAL | Disable Macro Expansion Listing           |
| MD    | Memory Display                            |
| MM    | Memory Modify                             |
| M     | "Alias" for previous command              |
| MS    | Memory Set                                |
| OF    | Offset Registers                          |
| PA    | Printer Attach                            |
| N0PA  | Printer Detach                            |
| PF    | Port Format                               |
| RD    | Register Display                          |
| RESET | Warm/Cold Reset                           |
| RM    | Register Modify                           |
| RS    | Register Set                              |
| SD    | Switch Directory                          |
| T     | Trace Instruction                         |
| TC    | Trace on Change of Flow                   |
| TM    | Transparent Mode                          |
| TT    | Trace to Temporary Breakpoint             |
| VE    | Verify S-Records                          |

#### Ukaz za vpis na pomnilniško lokacijo: MM

Vsebina pomnilniške lokacije je lahko podatek ali CPU ukaz. Oba lahko zapišemo v šestnajstki kodi. Pri ukazih je to njihova kodirani zapis, pri podatkih pa številčna vrednost. Po drugi strani je CPU ukaze elegantneje vpisati v mnemotehnični obliki.

Oba načina vpisa dosežemo z debug monitorskim ukazom **MM** (angl. memory modify), ki mu sledi naslov besede v RAM-u. Npr. ukaz

**MM 4000**

bo izpisal vsebino besede z naslovom  $4000_{HEX}$  (vsi naslovi so po definiciji zapisani v šestnajstki obliki) ter omogočil vpis nove vrednosti v šestnajstki kodi. Po zapisu nove vsebine in pritisku na tipko CR (ENTER) se izpiše vsebina naslednje lokacije in omogoči njen spremembu itd. Režim vpisa zapustimo tako, da vpišemo piko in CR (ENTER):

“prompt” debug  
monit. programa

|                    |                             |           |   |
|--------------------|-----------------------------|-----------|---|
| naslov<br>lokacije | stara vsebina v<br>HEX kodi | CPU32Bug> | <b>mm 4000</b><br>00004000 0123? 4444<br>00004002 660E? 4567<br>00004004 7949? 432 / nova vsebina<br>00004006 81A3? .<br>v HEX kodi |
|--------------------|-----------------------------|-----------|---|

Pri vpisovanju ukazov raje uporabimo opcijo **di** (angl. disassembler) za podpičjem, ki dezasembliira vsebino določenega naslova ter omogoči vpis novega ukaza v mnemotehnični obliki<sup>2</sup>:

```
CPU32Bug>mm 4000;di
00004000 4E71          nop
00004002 D280          add.l
00004004 043281A3 0411  SUBI.B   #\$A3,\$11(A2,D0.W*4) .
CPU32Bug>
```

Seveda je deasembliranje spominske lokacije smiselno le, če se na njej dejansko nahaja nek ukaz. V nasprotnem primeru bo rezultat dekodiranja ukaz, ki ni bil predviden na tej lokaciji. Debug monitor je vsebino lokacij od 4000 do 4008 iz prvega primera dekodiral kot ukaz

**SUBI.B #\\$A3,\\$11(A2,D0.W\*4).**

<sup>2</sup> Tako po vpisu novega ukaza se stara vsebina lokacije zbriše. Zato po pritisku na CR (ENTER) stara vsebina ni več vidna.

## Ukaz za izpis vsebine pomnilniške lokacije: MD

Ukaz izpiše vsebine nekaj spominskih lokacij (odvisno od opcije). Hkrati za vsako lokacijo izpiše tudi pomen vsebine v ASCII kodi (če kombinacija bitov sploh ustreza veljavnemu znaku):

```
CPU32Bug>md 4000
00004000 4E?1 D280 0432 81A3 0411 4504 FFFB 9C34      NqR..2.#..E..t.4
```

Lahko uporabimo tudi opcijo **di**, podobno kot pri ukazu MM:

```
CPU32Bug>md 4000;di
00004000 4E?1          NOP
00004002 D280          ADD.L    D0,D1
00004004 043281A3 0411 SUBI.B   #\$A3,\$11(A2,D0.W*4)
0000400A 4504          CHK.L    D4,D2
0000400C FFFB          DC.W     \$FFFFB
0000400E 9C340010      SUB.B    \$10(A4,D0.W),D6
00004012 84A887EF      OR.L     -\$7811(A0),D2
00004016 4EB09811      JSR      \$11(A0,A1.L)
CPU32Bug>
```

Hkrati z ukazi se izpišejo tudi kode ukazov (v HEX zapisu).

## Ukaza za izpis in spremembo vsebine registrov: RD in RM

Ukaza sta podobna prejšnjima ukazoma. V tem primeru izpišemo (**RD** - register display) ali sprememimo (**RM** - register modify) vsebine registrov CPU-ja (programskega števca - PC, statusnega registra - SR, podatkovnih in naslovnih registrov itd.). Primer ukaza **RD**:

```
CPU32Bug>rd
PC    =00003000  SR    =2700=TR:OFF_S_?.....  VBR    =00000000
SFC   =5=S0      DFC   =5=S0      USP   =0000FC00  SSP*   =00010000
D0    =00000000  D1    =00000000  D2    =00000000  D3    =00000000
D4    =00000000  D5    =00000000  D6    =00000000  D7    =00000000
A0    =00000000  A1    =00000000  A2    =00000000  A3    =00000000
A4    =00000000  A5    =00000000  A6    =00000000  A7    =00010000
00003000  0000F3B3      ORI.B    #\$B3,D0
CPU32Bug>
```

## Ukaz za zagon programa: GO (G)

Program, ki se nahaja v RAM ali ROM pomnilniku, poženemo z debug monitorskim ukazom **G0**, ki mu sledi naslov začetne lokacije. Npr. ukaz

```
G0 4000
```

bo pognal program, ki se nahaja na lokaciji 4000<sub>HEX</sub>. V nadaljevanju se izvajajo CPU ukazi v realnem času. Iz tega režima se nadzor vrne debug monitorju v naslednjih primerih:

- PC kaže na lokacijo, ki ne vsebuje veljavnega ukaza,
- vsebina PC-ja se izenači s prekinitveno točko (glej naslednja ukaza),

- program se izvaja v *trace* režimu,
- izvajanje prekinemo z RESET ali ABORT prekinitvijo (tipki na PFB-ju).

Ukaz **G0** brez naslova začne izvajati program na lokaciji, na katero trenutno kaže PC.

### Ukaz za vnos ali brisanje prekinitvene točke: **BR** in **NOBR**

Pri razvoju je včasih zelo koristno prekinjati izvajanje programa v izbranih točkah, ki jim rečemo *prekinitvene točke* (angl. breakpoints). Točke prekinitve določamo s pomočjo debug monitor ukaza **BR**, ki mu sledi naslov ukaza, pri katerem bo prišlo do prekinitve, npr.

**BR 4010**

Debug monitor omogoča definiranje do šestnajstih prekinitvenih točk. Po zagonu programa (ukaz **G0**) izvaja CPU ukaze v realnem času, vse dokler se vsebina PC-ja ne izenači z naslovom ene izmed prekinitvenih točk. Nato se nadzor vrne debug monitorju; hkrati se izpišejo vsebine registrov CPU. Od te točke naprej lahko program ponovno zaženemo z ukazom **G0** (brez definiranja naslova; glej prejšnji ukaz), do morebitne nove prekinitvene točke.

Posamezne prekinitvene točke “brišemo” z ukazom **N0BR** (angl. no breakpoint), kateremu sledi njen naslov. Ukaz brez naslova zbrisuje vse prekinitvene točke.

Primer:

V RAM-u se nahaja naslednji program z začetnim ukazom na lokaciji 3000<sub>HEX</sub>:

|          |          |      |        |           |
|----------|----------|------|--------|-----------|
| 00003000 | 203C0000 | 0123 | MOVE.L | #\$123,D0 |
| 00003006 | D440     |      | ADD.W  | D0,D2     |
| 00003008 | 02020001 |      | ANDI.B | #\$1,D2   |
| 0000300C | E556     |      | ROXL.W | #\$2,D6   |
| 0000300E | 4E?1     |      | NOP    |           |
| 00003010 | 00004284 |      | ORI.B  | #\$84,D0  |

Sledi definiranje prekinitvene točke in zagon programa:

```
CPU32Bug>br 3010
BREAKPOINTS
00003010
CPU32Bug>g 3000
Effective address: 00003000
At Breakpoint
PC    =00003010  SR    =2704=TR:OFF_S_?..Z..  VBR    =00000000
SFC   =5=S0      DFC   =5=S0      USP    =0000FC00  SSP*   =00010000
D0    =00000123  D1    =00000000  D2    =00000101  D3    =00000000
D4    =00000000  D5    =00000000  D6    =00000000  D7    =00000000
A0    =00000000  A1    =00000000  A2    =00000000  A3    =00000000
A4    =00000000  A5    =00000000  A6    =00000000  A7    =00010000
00003010 00004284          ORI.B      #$84,D0
CPU32Bug>
```

### Ukaz za zasledovanje: **T**

Prekinitvene točke omogočajo trenutno prekinjanje programa le v izbranih točkah. Včasih pa je zelo koristno opazovati posledice ysakega ukaza na stanje registrov. To nam omogoča ukaz T (angl. trace - zasledovanje). Z njim se izvrši le CPU ukaz, na katerega kaže PC. Pritisn na tipko CR (ENTER) ponovi izvajanje debug monitorskega ukaza T, le da sedaj kaže PC že na naslednji CPU ukaz. Z vsakokratnim pritiskom na tipko CR se bo ukaz T ponovil:

```
CPU32Bug>t
PC =00003006 SR =2700=TR:OFF_S_?..... VBR =00000000
SFC =5=$D DFC =5=$D USP =0000FC00 SSP* =00010000
D0 =00000123 D1 =00000000 D2 =00000000 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00010000
00003006 D440 ADD.W D0,D2
CPU32Bug>
PC =00003008 SR =2700=TR:OFF_S_?..... VBR =00000000
SFC =5=$D DFC =5=$D USP =0000FC00 SSP* =00010000
D0 =00000123 D1 =00000000 D2 =00000123 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00010000
00003008 02020001 ANDI.B #$1,D2
CPU32Bug>
PC =0000300C SR =2700=TR:OFF_S_?..... VBR =00000000
SFC =5=$D DFC =5=$D USP =0000FC00 SSP* =00010000
D0 =00000123 D1 =00000000 D2 =00000101 D3 =00000000
D4 =00000000 D5 =00000000 D6 =00000000 D7 =00000000
A0 =00000000 A1 =00000000 A2 =00000000 A3 =00000000
A4 =00000000 A5 =00000000 A6 =00000000 A7 =00010000
0000300C E556 R0XL.W #$2,D6
CPU32Bug>
```

### Ukaz za nalaganje datotek S19 z nadrejenega računalnika: L0

Ukaz L0 (angl. load - naloži) uporabljam za nalaganje datotek v S19 formatu, ki smo jih ustvarili v "off-line" režimu (pogl. 9) z nadrejenega računalnika v RAM pomnilnik mikrokrmlnika. Postopek je naslednji:

- z ukazom L0 pripravimo BCC na sprejem S19 datoteke,
- s komunikacijskim programom na nadrejenem računalniku pošljemo datoteko S19 prek serijskih vrat,
- po končanem prenosu pritisnemo dvakrat tipko CR (ENTER); debug monitor prevzame nadzor.

### 8.3 Komunikacija med terminalom (PC-jem) in BCC-jem

Uporabnik v “on-line” režimu komunicira z debug monitorjem BCC-ja prek PC ali Apple Macintosh računalnika<sup>3</sup>. Pri tem morata biti izpolnjena dva pogoja (glej tudi pogl. 6.3 o SCI):

- serijska vrata (SCI) BCC-ja morajo biti konfigurirana (nalogi debug monitorja) in
- računalnik mora delovati kot terminal.

Slednje dosežemo s pomočjo programov za emulacijo (posnemanje) terminala, kot so npr. ProComm, Kermit, Hyper Terminal itd., za PC kompatibilne računalnike in npr. Mac Terminal za Apple Macintosh računalnike. V terminalskem programu moramo nastaviti takšne parametre, ki ustrezajo konfiguraciji SCI porta na BCC-ju. Po definiciji veljajo naslednji parametri (izkušeni programerji lahko te parametre spremenijo z reprogramiranjem EPROM pomnilnika):

- hitrost prenosa: 9600 bps,
- 8 bitov na znak,
- pariteta: ne,
- stop biti: 1.

---

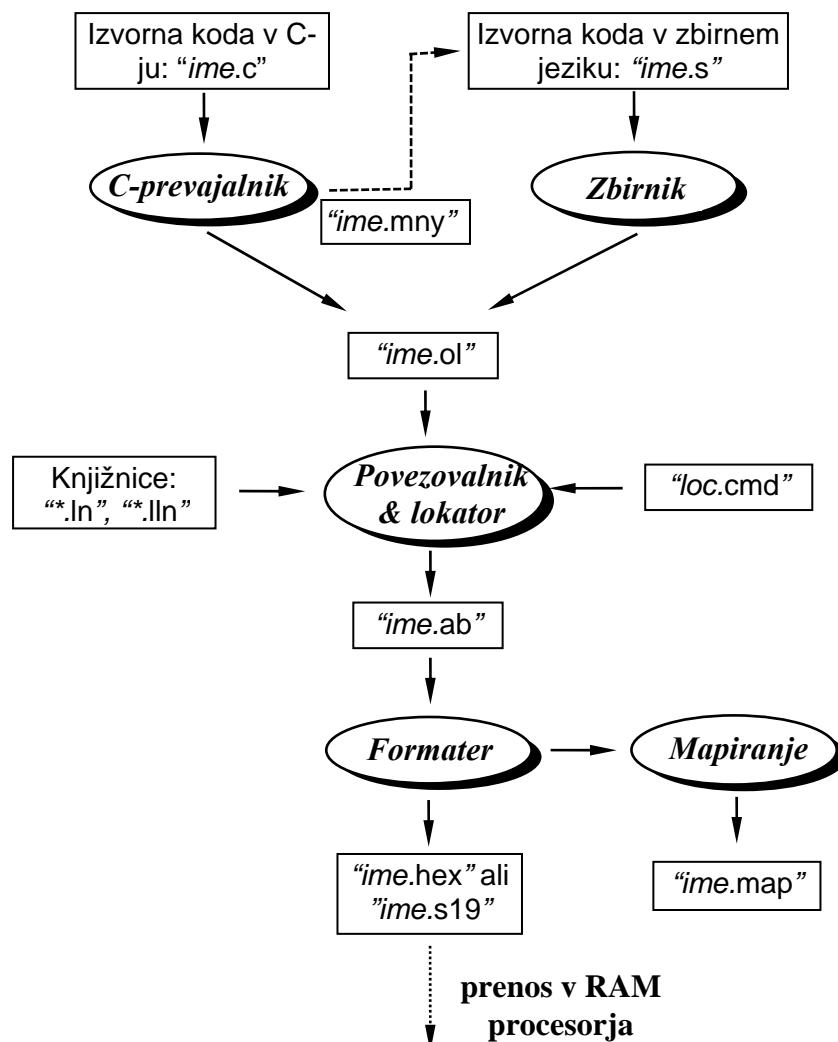
<sup>3</sup> Na istem računalniku lahko delamo tudi v “off-line” režimu, na BCC se priključimo pri prenosu S19 datoteke in zagonu ali testiranju programa.

## 9. RAZVOJNA PROGRAMSKA OPREMA

V predhodnem poglavju smo si ogledali dva osnovna načina pisanja programov za mikroprocesorske sisteme:

- neposredno na mikrokrilmilniku (on-line) in
- neodvisno od procesorja (off-line) na osebnem ali kakem drugem računalniku.

V tem poglavju bomo pokazali programska orodja, ki so na razpolago pri pisanju in analiziranju programov, ki jih pišemo v "off-line" režimu. Čeprav se sestava softverske opreme različnih proizvajalcev nekoliko razlikuje, so si koraki celotne procedure zelo podobni. Na sliki 9. 1 je prikazan del programske opreme firme "Intertools" za Motoroline mikroprocesorje serije 68XXX [33].



*Slika 9. 1: Pomožna programska oprema za delo z mikroprocesorjem*

## 9.1 Prevajanje programa

Ne glede na to, ali se odločimo za pisanje programa v jeziku C ali v zbirniku, je postopek zelo podoben. V prvi fazi napišemo na osebnem računalniku t.i. "izvorni program" ali "izvorno kodo" (angl. source code). Tak program pišemo v navadnem urejevalniku besedil in ga tudi shranimo v obliki navadnega teksta (ASCII tekst)<sup>1</sup>. Izvorni program mora ustrezati vsem leksikografskim pravilom, ki jih C ali zbirnik poznata. Slika 9. 2 kaže primer zelo enostavnega programa v jeziku C.

```
/* Primer enostavnega izvornega programa v C jeziku */
int a=1,b,c,d;          /* deklaracija in inicializacija
                           spremenljivke */
main()                   /* glavni program */
{
    c = a + b;           /* sestevanje spremenljivk */
    d = 7 * b;           /* mnozenje neinicializirane
                           spremenljivke s konstanto*/
}
/* konec programa */
```

*Slika 9. 2: Primer izvornega programa v C jeziku*

Naslednji korak je prevajanje v “*objektno kodo*” (angl. object code; datoteka s podaljškom “.ol”). Če je izvorni program pisan v C jeziku, ga prevajamo s pomočjo *C prevajalnika* (angl. compiler; od tod tudi žargonski izraz “kompajliranje”), iz zbirnega jezika pa ga prevajamo s pomočjo *zbirnika - asemblerja* (angl. assembling). Objektna koda je sestavljena iz kod CPU ukazov, ki smo jih obravnavali v poglavju 4. Poleg tega se v njej nahajajo še nekateri dodatni podatki, ki so potrebni pri nadaljnji obdelavi objektne kode. Objektno kod za izvorni program s slike 9. 2 kaže slika 9. 3. Sestavljena je iz segmentov, katerih pomen je nezahtevnemu programerju nerazumljiv. Z ozirom na to, da je to le vmesni korak, se na tem mestu ne bomo ubadali z razlagajo posameznih delov datoteke.

Na sliki smo posebej označili kodo ukaza **MOVE.L D0,D1**. Ta je del skupine ukazov v zbirnem jeziku, ki ustreza C ukazu "`c = a + b;`" (glej sliko 9.4 in pogl. 9.1.1). V tej fazi mu še ni dodeljena konkretna lokacija v pomnilniku.

<sup>1</sup> Večina orodij je predvidena za delo na preprostih operacijskih sistemih (praviloma MS-DOS), kar omogoča uporabo tudi na manj zmogljivih osebnih računalnikih. Najlažje in “najelegantnejše” je delo s čim preprostejšimi urejevalniki teksta, npr. “Edit”, itd.

```

○ .id "target" "68020"
○ .id "translator" "as68k.exe"
○ .id "date" "Nov 18 1999 17:37:23"
○ .id "as68k.exe sid" "@(#)as68k.PL 3.69.1.1"
○ .seg {PSCT} %1 1 1 {}
○ .seg {idata} %2 1 1 {data}
○ .seg {udata} %4 1 1 {data}
○ .seg {S_main} %8 1 1 {code}
○ .len %1 0
○ .org %1 0
○ .len %2 4
○ .org %2 0
○ .defg {_a} %3 %2 #0 +
○ '00000001'
○ .len %4 12
○ .org %4 0
○ .defg {_b} %5 %4 #0 +
○ 0:W 2
○ .defg {_c} %6 %4 #4 +
○ 0:W 2
○ .defg {_d} %7 %4 #8 +
○ 0:W 2
○ .len %8 28
○ .org %8 0
○ .ext {__main}%9
○ .defg {__main} %10 %8 #0 +
○ '222d'
○ 0 %2 + {data}%11 -:I
○ 'd2ad'
○ 0 %4 + %11 -:I
○ '2b41'
○ 4 %4 + %11 -:I
○ '202d' koda ukaza
○ 0 %4 + %11 -:I MOVE.L D1,D0
○ '2200'
○ 'e788'
○ '9081'
○ '2b40'
○ 8 %4 + %11 -:I
○ '4e75'
○ .group {data}%11 1 65535 %2 %4
○
○

```

*Slika 9. 3: Objektna koda*

Pri prevajanju imamo na voljo različne opcije. Zanimiva je opcija optimiranja. Program lahko namreč optimiramo z ozirom na pomnilniški prostor ali čas. To pa pomeni, da ga lahko poskušamo prevesti v najkrajšo možno obliko (s tem prihranimo na pomnilniškem prostoru) ali pa poskušamo dobiti program, ki se bo izvajal v čim krajšem možnem času. V nasprotju s pričakovanji najkrajša objektna koda ni nujno tudi najhitrejša, o čemer se bomo prepričali v naslednjem poglavju.

### **9.1.1 Prevajanje iz C izvornega programa v “psevdobirnik”**

Objektno kodo dobimo neposredno s prevajanjem izvornega programa iz C jezika ali zbirnika. Večasih pa si je zanimivo ogledati, kako bi izgledal program, ki smo ga napisali v C

jeziku, če bi ga pisali v zbirnem jeziku s CPU ukazi. Zato je pri večini prevajalnikov predvidena tudi ta opcija<sup>2</sup>. Slika 9. 4 kaže rezultat tega postopka (datoteka s podaljškom “.mny”).

```

○      SECTION     idata,, "data"
○      XDEF    _a
○  _a  DC.L   1
○
○      SECTION      udata,, "data"
○      XDEF    _b
○  _b  DCB    2,0
○      XDEF    _c
○  _c  DCB    2,0
○      XDEF    _d
○  _d  DCB    2,0
○ *1  /* Primer enostavnega izvornega programa v C kodi */
○ *2
○ *3  int a=1,b,c,d;           /* deklaracija in inicializacija
○ spremenljivk */
○ *4
○ *5  main()                  /* glavni program */
○
○      SECTION      S_main,, "code"
○      _bringin  __main
○      XDEF    _main
○  _main
○
○ *6  {
○ *7      c = a + b;          /* sestevanje spremenljivk */
○      MOVE.L    _a-data(A5),D1
○      ADD.L    _b-data(A5),D1
○      MOVE.L    D1,_c-data(A5)
○ *8
○ *9      d = 7 * b;          /* mnozenje neinicjalizirane
○ spremenljivke
○      MOVE.L    _b-data(A5),D0
○      MOVE.L    D0,D1
○      LSL.L #3,D0
○      SUB.L    D1,D0
○      MOVE.L    D0,_d-data(A5)
○ *10     s konstanto*/
○ *11
○ *12  }                      /* konec programa */
○      RTS

```

*Slika 9. 4: Program v C-ju s slike 9. 2 v mnemotehnični obliki (zbirni jezik)*

C ukazi so tukaj zapisani v obliki oštevilčenih komentarjev. Programerju je vsekakor zanimiv pogled na komentirani C ukaz s številko \*9 za množenje s konstanto. Prevajalnik je namesto CPU ukaza (npr.) MNYΣ uporabil nekaj nadomestnih ukazov; množenje s konstanto 7 smo dosegli tako, da smo originalno spremenljivko (v registrih D0 in D1) najprej pomaknili za tri bitna mesta na levo (množenje z osem) in nato od rezultata odšteli originalno vrednost.

<sup>2</sup> Ta korak je pri navadnem prevajanju popolnoma nepotreben in ponavadi služi le v informativne namene.

Nadomestna sekvenca se bo izvršila hitreje od navadnega množenja (podrobnejši opis je podan v pogl. 4.4).

### 9.1.2 Elementi zbirnega jezika

V poglavju 4 smo našteli vse ukaze CPU32 v mnemotehnični obliki, ki so osnovni del vsakega programa v zbirnem jeziku. Poleg njih vsebuje program še nekaj skupin pomožnih instrukcij (t.i. psevdoukazov - angl. pseudo ops.)<sup>3</sup>, ki jih uporabljamo za organizacijo uporabniškega programa in se zato ne prevajajo v objektno kodo (razen ukaza DC). Tukaj bomo našteli le nekaj značilnih ukazov zbirnika. Nekatere bomo uporabili v Poglavlju 10.

#### 9.1.2.1 Krmiljenje zbirnika

- END je ukaz zbirnika s katerim zaključimo uporabniški program. Zbirnik ignorira vse naslednje ukaze.
- Z ukazom INCLUDE lahko v izvorni program vključimo neko sekundarno datoteko (npr. datoteko s tabelo sinusnih vrednosti).
- Ukaz ORG (angl. absolute origin) spremeni vsebino programskega števca. Z njim lahko definiramo dejanski začetek uporabniškega programa.

#### 9.1.2.2 Določanje simbolov

S temi ukazi simboličnim nazivom dodelimo konkretnne vrednosti.

- EQU (angl. Equate Symbol Value) dodeli številčno vrednost simbolu (npr. ukaz k1 equ \$50 bo simbolu k1 dodelil vrednost 50<sub>HEX</sub>).
- ukaz SET je podoben prejšnjemu ukazu, razlika je edino v tem, da lahko pri njem med izvajanjem programa spremenimo številsko vrednost dodeljenega simbola, kar pri ukazu EQU ni dovoljeno.

#### 9.1.2.3 Definiranje podatka in dodeljevanje pomnilniškega prostora

S temi ukazi rezerviramo pomnilniški prostor, v katerem bomo vnesli podatke.

- Z ukazom DC.x (angl. Define Constant; x = B, W, L itd.) bomo v spominu rezervirali prostor določene velikosti in v njega vpisali neko konstanto. Npr.

**DC.W 10,5,7 \*inicijaliziraj tri 16-bitne lokacije**

Zaporedna ukaza DC.B 'E' in DC.B 'J' bosta naložila dva zaporedna byta z vsebinami 45<sub>HEX</sub> in 4a<sub>HEX</sub>, to je z ASCII kodo črk E in J.

---

<sup>3</sup> Načeloma velja za zbirnike vseh proizvajalcev, v konkretnem zgledu pa je uporabljen zbirnik firme Intermetrics [33].

**PAZI!** Asemblerškega ukaza **DC.x** ne smemo mešati z ukazom **EQU**, saj slednji ne manipulira s pomnilniškim prostorom, temveč omogoča le preglednejše pisanje programa preko zamenjave številčne vrednosti s simbolom.

- Ukaz **DS.x y** (angl. Define Storage;  $x = B, W, L$  itd.,  $y$  je celo število) rezervira prostor v pomnilniški mapi. Ukaz je zelo pomemben pri povezovanju, saj določa področje, na katerega ni možno shranjevati ukazov, podatkov, spremenljivk itd. Primer:

```
lok      DS.W $10 *rezerviraj prostor 1b(DEC) besed začenši
z
*lokacijo lok.
```

### 9.1.3 Formatiranje izpisa

V tej skupini so ukazi, ki formatirajo izpis programa v .lst datotekah in nimajo nobenega vpliva na sam uporabniški program. Primer: po ukazu **PAGE** se bo izpis nadaljeval na naslednji strani.

### 9.1.4 Nadzor nad zunanjimi simboli

Osnovno pravilo pri pisanju programov je, da enakega simboličnega imena ne smemo dodeliti več spremenljivkam. V naslednjem poglavju o povezovanju bomo omenili možnost modularnega pisanja delov in njihovega združevanja v celoto. Pri tem pristopu bomo uporabljali nekatere spremenljivke, ki smo jih definirali v drugih modulih pred združitvijo. Pozorni moramo biti na dve možni napaki:

- prevajalnik izvrši svojo funkcijo brez napak le, če so vsi simboli za spremenljivke, ki jih uporabljammo v določenem modulu, na začetku tega modula tudi deklarirane,
- povezovalnik ne dovoljuje večkratnega deklariranja simbolov spremenljivk v modulih, ki jih bomo združili v eno celoto, čeprav so bili pisani ločeno.

Zato moramo vsako simbolično podano veličino definirati v enem modulu (ukaz **XDEF** - angl. external definition), v drugih modulih, ki ta simbol uporabljammo, pa na začetku programa z ukazom **XREF** (angl. external reference) prevajalniku javiti, da bo simbol definiran v drugem modulu.

## 9.2 Povezovanje

Povezovanje (“linkanje”, angl. linking) je zelo pomemben korak do končnega uporabnega programa. Tukaj bomo omenili le nekatere funkcije povezovanja:

- povezovanje z ostalimi objektnimi moduli,
- povezovanje s knjižnicami,
- določanje končne lokacije programa v ciljnem programu.

Datoteka “\*.ol” je rezultat prevajanja le enega programa ali programskega modula. Ta modul bi lahko bil del večjega programa, ki vsebuje še druge podobne module. Povezovanje v skupno celoto opravimo s povezovalnikom.

Nekatere funkcije (poenostavljeni: ukazi) v C-ju so zelo zapletene in sestavljene iz enostavnnejših ukazov. Programerju so te funkcije na voljo v t.i. knjižnicah (angl. libraries). Knjižnice so urejene po sorodnosti funkcij: npr. standardne vhodno/izhodne funkcije (izpis znakov na zaslon, branje s tipkovnice, itd.), matematične funkcije (npr. trigonometrične in logaritmične funkcije), pretvorbe formatov itd. Programer lahko tudi sam definira uporabniške knjižnice, ki jih po potrebi vključuje v program. Knjižnice moramo pri povezovanju obvezno dodati k objektni kodi izvornega programa.

Objektna koda ne vsebuje informacije o naslovih na katerih se bo nahajal končni program v RAM ali ROM pomnilniku procesnega mikroračunalnika, kar je tudi smiselno, saj ga moramo še povezati z dodatnimi moduli in knjižnicami. Končne lokacije programskih segmentov določa povezovalnik s pomočjo posebne datoteke (v našem primeru "loc.cmd", slika 9. 5).

```
MEMORY (#10000);           -- M68332 RAM je omejen na 64 K bytov.  
RESERVE (#0 TO #3000);     -- Rezerviranje spodnjega področja RAM-a za  
                           -- lastne potrebe  
LOCATE (init : #3000);     -- Zacetek uporabniškega programa (kode)  
LOCATE({code} {} {constant} {data} {isep} {usep} {stsep} S_end_project: #3080);  
                           -- Lociranje posameznih segmentov programa
```

*Slika 9. 5: Primer datoteke loc.cmd*

V tej preprosti datoteki smo opisali pomnilniško konfiguracijo ciljnega procesnega mikroračunalnika (velikost RAM pomnilnika 64 Kbyte =  $10000_{\text{HEX}}$ , pri čemer je spodnje področje rezervirano). Začetek programa se nahaja na lokaciji  $3000_{\text{HEX}}$ .

V vsakem programu lahko ločimo posamezne segmente z nekaterimi skupnimi lastnostmi. To so npr. *ukazi* (ali koda - angl. code), *konstante* (angl. constants, npr. tabelarično podani podatki), *inicjalizirane* ali *neinicjalizirane spremenljivke* itd. Prevajalnik identificira pripadnost posameznih delov izvornega programa tem segmentom in jih tudi grupira. Pri povezovanju imamo možnost izbire naslovov segmentov znotraj dostopnega pomnilniškega prostora, kar storimo z zadnjo vrstico v datoteki "loc.cmd". To je pomembno zlasti, če bomo končni program "zapekli" v (E)EPROM pomnilnik<sup>4</sup>. V konkretnem primeru nima niti eden izmed segmentov s slike 9. 5 vnaprej določene začetne lokacije, torej jih bo povezovalnik samostojno porazdelil po dostopnem pomnilniškem prostoru, hkrati pa bo programerju posredoval informacije o tem (glej poglavje o mapiranju).

<sup>4</sup> V ROM tipu pomnilnika lahko shranimo ukaze in konstante, ne pa tudi npr. neinicializiranih spremenljivk. Naslov njihovega segmenta moramo določiti v nekem RAM pomnilniku.

Končni rezultat povezovanja je datoteka s podaljškom imena “.ab” (njen del prikazuje slika 9. 6). Kot vidimo, so tukaj že določene končne lokacije posameznih segmentov ter njihove velikosti. Koda ukaza **MOVE.L D0,D1** je del skupine ukazov, ki se začne na naslovu **31BеHEx** (asemblerSKI ukaz **.aorg %3 #31Bе**). Preostali del datoteke vsebuje kode ukazov in podatke, po svoji obliki pa je zelo podoben objektni kodi s slike 9. 3.

```

○ .id "date" "Nov 18 1999 17:33:39"
○ .id "target" "68020"
○ .id "llink sid" "@(#)llink.PL 1.109.1.1"
○ .seg {idata}%1 1 u {data}
○ .len %1 #c
○ .abs %1 #318a
○ .seg {udata}%2 1 u {data}
○ .len %2 #8c
○ .abs %2 #3196
○ .seg {_main}%3 1 u {code}
○ .len %3 #1c
○ .abs %3 #316e
○ .seg {init}%4 1 u {code}
○ .len %4 #48
○ .abs %4 #3000
○ .seg {_exit_1}%5 1 u {code}
○ .len %5 #28
○ .abs %5 #3146
○ .seg {_atexit}%6 1 u {code}
○ .len %6 #56
○ .abs %6 #30f0
○ .seg {_end_project}%7 1 u {endall}
○ .len %7 4
○ .abs %7 #3222
○ .seg {_alloc}%8 1 u {code}
○ .len %8 #70
○ .abs %8 #3080
○ .group {data}%9 1 152 %1 %2
○ .id "translator" "llink"
○ .defg {ldata}%10 #98
○ .defg {_a}%11 #318a
○ .defg {_b}%12 #3196
○ .defg {_c}%13 #319a
○ .defg {_d}%14 #319e
○ .defg {_main}%15 #3000
○ .defg {_main}%16 #316e
○ .defg {_main1}%17 #3156
○ .defg {_exit1}%18 #3044
○ .defg {_al_init}%19 #30d2
○ .defg {_exit}%20 #3114
○ .defg {_exit_1}%21 #3146
○ .defg {_exit}%22 #314e
○ .defg {_atexit}%23 #30f0
○ .defg {_end_project}%24 #3222
○ .defg {_alloc}%25 #3080
○ .start #3000
○ .id "as68k.exe sid" "@(#)as68k.PL 3.69.1.1"
○ .id "mod" "test.ol"
○ .aorg %1 #318a
○ '00000001'
○ .aorg %2 #3196
○ 0:w 2
○ 0:w 2
○ 0:w 2
○ .aorg %3 #316e
○ '222d'
○ 0:I
○ 'd2ad'
○ '#c:I
○ '2b41'
○ '#10:I
○ '202d'
○ '#c:I
○ '2200' → koda ukaza
○ e788
○ '9081'
○ '2b40'
○ '#14:I
○ '4e75'
○ .id "llink sid" "@(#)llink.PL 1.109.1.1"
○ .id "as68k.exe sid" "@(#)as68k.PL 3.69.1.1"
○ .id "mod" "pmn340b.ol"
○ ...

```

koda ukaza  
MOVE.L D0,D1

Slika 9. 6: Del programa s slike 9. 2 po povezovanju (podaljšek „.ab”)

### 9.3 Formatiranje

Vsi prej opisani postopki so imeli za cilj ustvariti končno datoteko s programom, ki ga želimo vstaviti v pomnilnik našega mikroračunalniškega sistema. Pri tem lahko program naložimo v:

- ROM (običajno EPROM) ali v
- RAM pomnilnik.

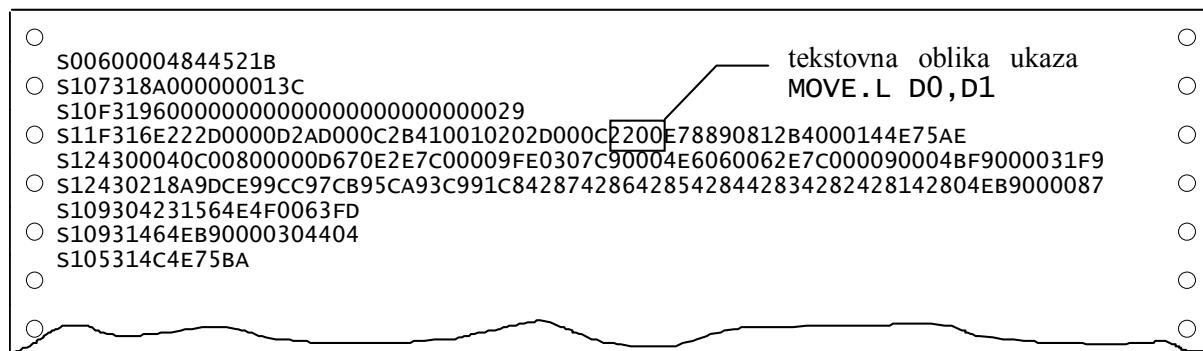
Prva varianca pride največkrat v poštev le, ko smo prepričani, da deluje program v skladu s pričakovanji in ga nimamo namena naknadno spremnjati. Takrat lahko program prilagodimo za takojšnje delovanje ob vsakokratnem vklopu mikroračunalnika.

V razvojni fazi je druga inačica veliko bolj elegantna. Ideja je v tem, da končni program z računalnika, na katerem smo program razvili (angl. host computer, najpogosteje osebni računalnik), naložimo v RAM pomnilnik procesnega mikroračunalniškega sistema in ga po potrebi spremnjamo ali ponovno naložimo.

Prenos se običajno vrši preko serijskega protokola EIA-232 za prenos ASCII znakov (glej poglavje 6.3). Datoteka iz prejšnjega podpoglavlja, ki je rezultat povezovanja, ni primerna za takšen prenos, zato so proizvajalci mikroprocesorskih komponent (npr. Motorola, Texas Instrument, Hewlett Packard...) ustvarile lastne formate za prenos.

Motorola uporablja S19 format (datoteke s podaljškom “.s19” ali “.hex”), ki je prikazan na sliki 9. 7. Za pretvorbo dokončne objektne kode (podaljšek “.ab”) v tekstovni S19 format skrbi posebni program, t.i. *formater* (angl. formatter).

Na sliki smo ponovno označili kodo ukaza **MOVE .L D0 ,D1** v tekstovni (ASCII) obliki.



**Slika 9. 7: Del datoteke z S19 formatiranim programom**

Naziv Motorolinega formata izhaja iz začetnih črk vsake vrstice (od S0 do S9), ki določajo pomen teh vrstic. Npr., vrstica S0 lahko vsebuje opisne informacije o programu, S9 pa zaključuje, vrstice z začetnicami S1, S2 in S3 pa vsebujejo podatke ali kode ukazov. Razlika med vrsticami S1, S2 in S3 je le v formatu naslova, na katere se shranjujejo podatki ali kode:

pri S1 je naslov sestavljen iz dveh bytov, pri S2 iz treh, pri S3 pa iz štirih. Oglejmo si primer zadnje vrstice:

|      |   |  |
|------|---|--|
| S1   | - | oznaka vrstice   |
| 05   | - | kontrolno število (v HEX formatu), ki pove koliko bytov mu sledi v vrstici |
| 314C | - | naslov   |
| 4E75 | - | koda ukaza ali podatek   |
| BA   | - | kontrolna vsota (angl. checksum)   |

Kontrolna vsota je le podatek, ki vsebuje informacijo o pravilnosti prenosa podatkov. To je byte z najmanjo težo enojnega komplementa vsote vseh bytov (brez uvodnih bytov Sn).

V našem primeru:

$$05_{\text{HEX}} + 31_{\text{HEX}} + 4C_{\text{HEX}} + 4E_{\text{HEX}} + 75_{\text{HEX}} = 145_{\text{HEX}},$$

$$\sim 145_{\text{HEX}} = \text{eba}_{\text{HEX}},$$

kjer upoštevamo le byte z najnižjo težo  $\text{ba}_{\text{HEX}}$ .

Sprejemnik (monitorski program v procesnem mikroracunalniku) na koncu vsake vrstice izračuna kontrolno vsoto in jo primerja z zadnjim sprejetim bytom.

## 9.4 Mapiranje

Pri pisanju programskega modula v C jeziku ali zbirniku programer načeloma nima vpogleda v naslove, na katerih se bodo nahajali spremenljivke, procedure in konstante (razen če jih sam eksplicitno ne definira). V fazi testiranja programa v procesnem računalniku pa si pogosto želimo ogledati vsebine pomnilniških področij, na katerih se nahajajo spremenljivke ali podatki, ali locirati določen podprogram v pomnilniku zaradi sprotnega spremeljanja njegovega delovanja. *Mapiranje* (angl. mapping) je ena izmed neobveznih opcij, ki nam omogoča izpis vseh relevantnih podatkov o položaju posameznih segmentov programa v pomnilniku (angl. cross-reference). Rezultat mapiranja (datoteka z podaljškom ".map") je zelo zanimiv, saj nam veliko pove o konceptu povezovanja in upravljanja s pomnilnikom, zato jo prikažimo v celoti (slika 9. 8).

Skupina ukazov s slike 9. 2, katere del je tudi ukaz **MOVE.L** **D0,D1** se nahaja v sekiji **\_main**, ki se nahaja na naslovu  $316E_{\text{HEX}}$ . Na sliki 9. 2 so to ukazi v glavnem programu (**main**), od C ukaza **c = a + b;** naprej.

```

○
○ Translator : llink
○ Target      : 68020
○ Global      Address
○ __main     00003000 (12288)
○ __exit1    00003044 (12356)
○ __alloc     00003080 (12416)
○ __al_init   000030d2 (12498)
○ _atexit    000030f0 (12528)
○ _exit      00003114 (12564)
○ __exit_1   00003146 (12614)
○ __exit     0000314e (12622)
○ __main1    00003156 (12630)
○ _main      0000316e (12654)
○ _a         0000318a (12682)
○ _b         00003196 (12694)
○ _c         0000319a (12698)
○ _d         0000319e (12702)
○ _end_project 00003222 (12834)
○ Group      Size Limit      Align Member Segments
○ data       000098 (152)      hword idata      udata
○ Segment    Address          Length   Class   Align   Combine
○ init       00003000 (12288)  000048 (72)  code    hword  private
○ S_alloc    00003080 (12416)  000070 (112) code    hword  private
○ S_atexit   000030f0 (12528)  000056 (86)  code    hword  private
○ S_exit_1  00003146 (12614)  000028 (40)  code    hword  private
○ S_main     0000316e (12654)  00001c (28)  code    hword  private
○ idata      0000318a (12682)  00000c (12)  data    hword  private
○ udata      00003196 (12694)  00008c (140) data    hword  private
○ S_end_project 00003222 (12834) 000004 (4)  endall hword  private
○
○ Statistics
○ Segments   : 8
○ Globals    : 16
○ Groups     : 1
○ Sum of class "code" segments : 00000152 (338)
○ Sum of class "data" segments : 00000098 (152)
○ Sum of all other segments   : 00000004 (4)
○ Total size of all segments : 000001ee (494)
○ User Start Address = #3000
○

```

Slika 9. 8: Datoteka ".map"

## 10. PRIMER UPORABE MIKROKRMILNIKA

Velika večina mikroprocesorjev, ki so predvideni za manj in srednje zahtevne aplikacije, uporablja le celoštevilski (`integer` v C jeziku) format (npr. 8-bitni Motorola MC68HC11, 16-bitni DSP Texas Instruments 320F240 ali 32-bitni MC68332). Mikroprocesorje z vgrajenim ali dodatnim matematičnim koprocesorjem, ki omogoča bistveno hitrejše računanje s števili zapisanimi v formatu s plavajočo vejico, najdemo šele v višjem cenovnem razredu za zahtevnejše aplikacije, čeprav jim tudi tu konkurirajo sodobni mikroprocesorji brez koprocesorja.

Ena od prednosti procesorjev z matematičnim koprocesorjem je enostavnejša manipulacija s spremenljivkami v programu. Aritmetika s plavajočo vejico omogoča neposredno računanje z realnimi števili, ki lahko ustrezajo vrednostim fizikalnih veličin. Npr. toku 5,87 A v C jeziku lahko priredimo `float` ali `double` spremenljivko `tok = 5,87`. Množenje te spremenljivke s `float` spremenljivko `upornost = 2,3 (Ω)` bo dalo rezultat 13,501, ki ustreza napetosti (npr. `float` spremenljivka `napetost`). Pri tem ne smemo pozabiti, da imajo vhodne (A/D pretvorniki) in izhodne (D/A pretvorniki) enote računalnika na digitalni strani le celoštevilske vrednosti.

Celoštevilska aritmetika izračuna iz predhodnega odstavka ne omogoča na tako preprost način. Eden načinov rešitve tega problema bi lahko bila prilagoditev celoštevilskih spremenljivk vrednostim fizikalnih veličin. Npr. toku 5,87 A priredimo (`short int`, ali `long int`) spremenljivko `tok = 5870`, upornosti pa dodelimo vrednost 2300, torej fizikalne vrednosti množimo s faktorjem 1000 oz. zaokrožimo na tretjo decimalko. Seveda bo treba rezultat izračuna (13501000) deliti z  $10^6$ , da dobimo ustrezno prilagojeno vrednost. Poleg dejstva, da uporabljajo računalniki binarno logiko (torej zahteva deljenje z decimalno vrednostjo  $10^6$  dodatne ukaze), je očitno tudi, da na ta način ne moremo popolnoma izkoristiti ločljivosti mikroprocesorja (npr.  $2^{16}$  ali  $2^{32}$ ). Enako ostaja vprašljivo, kako bi zapisali velika števila, ki presegajo maksimalno vrednost v registru (npr. 1234,939 oz. 1234939 v 16-bitnem registru). Zaradi tega se pri podobnih operacijah raje poslužujemo normiranja.

### 10.1 Normiranje fizikalnih veličin

V tem poglavju bomo opisali enega izmed načinov normiranja veličin ter ustrezne matematične operacije. V tem in nadaljnjih primerih bomo predpostavili, da je dolžina osnovne besede 16 bitov. To je ločljivost, ki zadošča večini srednje zahtevnih in nekaterim zahtevnejšim nalogam.

Z normiranjem maksimalno izkoristimo številski okvir, ki ga ponuja neki CPU. Običajno moramo pri tem upoštevati logiko dvojiškega komplementa, saj imamo opravka s pozitivnimi in negativnimi vrednostmi spremenljivk<sup>1</sup>. Maksimalni pozitivni vrednosti pri 16-bitni CPU bo

---

<sup>1</sup> Izjema so nekatere veličine, ki jim lahko dodelimo le pozitivne vrednosti. Npr. kot rotorja motorja lahko vedno zapišemo s pozitivnimi vrednostmi od 0 do  $2\pi$ , kar bi ustrezalo številskem področju od 0000<sub>HEX</sub> do ffff<sub>HEX</sub> (v 16-bitnem sistemu). Negativne kote lahko vedno ponazorimo s pozitivnimi.

potemtakem lahko ustrezala vrednost  $7\text{fff}_{\text{HEX}}$  ( $2^{15}-1 = 32767_{\text{DEC}}$ ), najmanjsa negativna vrednost pa je  $8000_{\text{HEX}}$  ( $-2^{15} = -32768_{\text{DEC}}$ )<sup>2</sup>.

*Normiranje* je postopek, v katerem neki vrednosti fizikalne veličine (npr. toku, napetosti, hitrosti) dodelimo referenčno številčno vrednost, s katero bo operiral procesor. Izbira norme je odvisna od veliko faktorjev: možnih minimalnih in maksimalnih vrednosti fizikalne veličine, ločljivosti, koherentnosti z normami ostalih veličin itd. Ponavadi pri normirjanju izberemo nazivno (npr. nazivni tok motorja) ali maksimalno možno vrednost (npr. kratkostični tok v motorju) fizikalne veličine. Katera heksadecimalna vrednost bo ustrezala normi v mikroprocesorju, je odvisno tudi od konkretnega problema, ki ga rešujemo. Seveda želimo številski obseg procesorja popolnoma izkoristiti, zato načeloma priredimo normi čim višjo vrednost. Na ta način bomo lahko računali tudi z vrednostmi, ki so bistveno manjše od normirane.

Norma običajno ustreza nazivni vrednosti, saj so tudi podatki o fizikalnih veličinah reguliranca (npr. motorja) podani za nazivno obratovanje (glej tudi zgled v pogl. 10.3.4). Pri tem pa je treba zelo paziti, da pri vrednostih fizikalnih veličin, ki so večje od norme, ne pride do presežka (overflow). Temu se lahko izognemo že pri izbiri norme, ki ustreza maksimalni vrednosti, to pa lahko pripelje do drugih nezaželenih posledic (npr. maksimalni vrednosti oz. normi toka ne ustreza nujno maksimalna vrednost ali norma napetosti).

Pri normirjanju na nazivno vrednost ustreza normi heksadecimalni ekvivalent, ki je manjši od maksimalne možne vrednosti (od  $-32768$  do  $+32767$ ). Na ta način se tudi pri nadnazivnih vrednostih fizikalne veličine izognemo presežku. Zgled v pogl. 10.3 pojasnjuje praktične probleme pri normirjanju.

Glede na to, da je človeku bližje računanje z realimi decimalnimi števili, običajno med preračunavanjem iz fizikalne vrednosti v heksadecimalno vpeljemo še pomožni zapis, kjer vrednost “-1” ustreza negativni vrednosti norme, “+1” pa pozitivni. Polovični vrednosti norme ustreza “+0,5” itd. (takšen prikaz bomo zaradi ločevanja od navadnega zapisa označevali z narekovaji).

### **Primer:**

Nazivna vrednost toka, ki teče v motor je 33 A, ki naj mu ustreza heksadecimalni ekvivalent  $4000_{\text{HEX}}$ . Tok 3 A je “0,0909” norme, torej:

$$33 \text{ A} \rightarrow “+1” \rightarrow 16384_{\text{DEC}} \rightarrow 4000_{\text{HEX}},$$

Šestnajstško vrednost, ki ustreza toku 3 A, pa dobimo takole:

$$3 \text{ A} \rightarrow “+0,0909” \rightarrow 0,0909 \cdot 16384_{\text{DEC}} \approx 1489_{\text{DEC}} \rightarrow 5d1_{\text{HEX}}.$$

Na ta način se izognemo neposrednemu računanju s šestnajstškimi števili; pretvorbo v ta format opravimo šele iz decimalnega zapisa.

<sup>2</sup> Pri 8-bitnih CPU-jih bi se vrednost gibala med  $80_{\text{HEX}}$  in  $7f_{\text{HEX}}$ , pri 32-bitnih pa med  $80000000_{\text{HEX}}$  in  $7fffffff_{\text{HEX}}$ .

S pravilno izbiro digitalnih ekvivalentov norm vseh nastopajočih veličin lahko popolnoma izkoristimo številski obseg procesorja. Dodatna prednost se kaže tudi v koherentnosti z veličinami, ki jih nismo neposredno normirali.

**Primer:**

Kot bomo videli v praktičnem zgledu, normiramo pri motorju tri osnovne veličine (napetost, tok in hitrost rotorja) na nazivne vrednosti, ki so podane na napisni tablici ali smo jih izmerili. S tem so enoumno podane tudi norme parametrov (npr. upornosti, induktivnosti, časovne konstante). Hkrati bodo tudi rezultati enačb normirani. Npr. fluks motorja izračunamo iz enačb, ki vsebujejo prej navedene vrednosti. Če je rezultat tega izračuna “0,7” pomeni, da je trenutno motor magneten s 70 % nazivnega fluksa.

### **10.1.1 Matematične operacije med normiranimi veličinami**

#### **10.1.1.1 Seštevanje in odštevanje**

Pri izvajanju operacij med normiranimi celoštivilskimi veličinami v mikroprocesorskem sistemu moramo biti zelo previdni. Enostavne matematične operacije, kot so npr. seštevanje in odštevanje ter logične povezave običajno potekajo brez težav. Edina resnejša nevarnost leži v možnosti preseganja področja osnovne besede.

**Primer:**

Zaradi enostavnosti predpostavimo, da so veličine normirane na maksimalno možno vrednost, torej “+1” ustreza  $+32767_{DEC}$  ( $7fff_{HEX}$ ), “-1” pa  $-32768_{DEC}$  ( $8000_{HEX}$ ). Rezultat seštevanja pozitivnih vrednosti  $5555_{HEX}$  (“ $0,667$ ” =  $21845_{DEC}$ ) in  $3000_{HEX}$  (“ $0,375$ ” =  $12288_{DEC}$ ), npr. z ukazom ADD, bo dalo  $8555_{HEX}$  (kar je v 16-bitnem računalniku negativna vrednost:  $-31403_{DEC}$ ), namesto  $+34133_{DEC}$  (“ $1,045$ ”)<sup>3</sup>. Dobljena vrednost je logična, saj bi pričakovani rezultat presegal maksimalno dovoljeno vrednost. Računalnik se bo odzval s setiranjem “overflow” bita (OV ali V) v statusnem registru. Previdnost je potrebna tudi zaradi tega, ker je potek izračuna binarnih zapisov povsem regularen in jo bo načeloma računalnik izvedel brez zapletov (razen postavljenega bita V).

#### **10.1.1.2 Množenje**

Pri operacijah množenja in deljenja je potrebna dodatna pozornost. Množenje dveh števil, ki sta manjši ali enaki normi ( $\leq “1”$ ), ima vedno kot rezultat število, ki je manjše ali enako vrednosti obeh vhodnih števil.

Npr. rezultat množenja normiranih števil iz prejšnjega primera bo:

$$“0,667” \cdot “0,375” = “0,25”. \quad (10. 1)$$

---

<sup>3</sup> Majhna razlika med seštevkom normiranih vrednosti in normiranim rezultatom je posledica zaokroževanja.

Poglejmo, ali sledi ta rezultat tudi kot posledica ukaza za 16-bitno množenje v 32-bitnem mikroprocesorju MC68332, npr. z ukazom **MULS·W D0,D2**. Operanda (spodnjih 16 bitov 32-bitnega registra) se nahajata v D0 in D2, rezultat (32 - bitni) pa v registru D2. Pri  $D0 = 5555_{HEX}$  in  $D2 = 3000_{HEX}$  bo rezultat  $D2 = ffff000_{HEX}$ , torej veliko večji od maksimalne vrednosti, ki jo dolžina besed operanda dovoljuje. Kako ta rezultat povezati s pričakovanim rezultatom v (10. 1)?

Množenje normiranih števil lahko zapišemo na naslednji način:

$$("0,667" \cdot 2^{15}) \cdot ("0,375" \cdot 2^{15}) = "0,25" \cdot 2^{30}.$$

Faktor  $2^{15}$  (in ne  $2^{16}$ , kot bi mogoče pričakovali) je namreč posledica dejstva, da smo z upoštevanjem dvojiškega komplementa (“-1” do “+1”) celotno področje  $2^{16}$  razdelili na dve polovici. Na ta način smo MSB uporabili za predznak, ki ne prispeva k ločljivosti. Če želimo, da bo rezultat pravilen, moramo eksponent postaviti na enako vrednost množenca in množitelja. V praksi to pomeni aritmetični pomik (angl. shift) rezultata za 15 bitov v desno (“odrežemo” spodnjih 15 bitov). V našem primeru se vsebina D2 spremeni v  $1fff_{HEX}$ . Po drugi strani velja iz (10. 1):

$$"0,25" \cdot 2^{15} = 8192_{DEC} = 2000_{HEX}.$$

Kot vidimo, sta rezultata praktično enaka. Do možne minimalne razlike pride iz dveh razlogov:

1. Normiranje oz. pretvarjanje iz realnih v cela števila nujno prinese nek pogrešek.
2. Pri normiranju (od “-1” do “+1”) smo predpostavili, da sta negativno in pozitivno območje simetrični. V resnici pa je maksimalna negativna celoštevilska vrednost vedno za 1 večja od maksimalne pozitivne zaradi prištevanja vrednosti “0” k pozitivnemu področju (glej pogl. 1.2).

Zgoraj opisana naloga je lažja, ko imamo opravka z množenjem dveh nepredznačenih (angl. *unsigned*) števil, ki se gibljeta v področju  $[“0”, “1”] \cdot 2^{16}$ . Takrat moramo iz rezultata  $[“0”, “1”] \cdot 2^{32}$  izločiti le spodnjih 16 bitov. Pri 16-bitnih CPU-jih je to še lažje doseči. Tam je produkt registrov sestavljen iz dveh zaporednih registrov. Rezultat bo avtomatično v tistem registru, ki vsebuje besedo zmnožka z višjo težo.

#### 10.1.1.2.1 Množenje z normiranimi vrednostmi, ki so večje od 1

Pri računanju z normiranimi števili pride včasih do potrebe po množenju s faktorjem, ki je večji od “1”, npr.

$$"0,12" \cdot "-3,65".$$

V primeru izbire norme, ki ustreza maksimalni vrednosti registra, zadnjega podatka ne moremo zapisati v enem samem registru, saj presega njegovo maksimalno območje. Zato je množenju treba pristopiti nekoliko drugače.

Zaradi binarne narave organizacije registrov je takšne podatke smiselno prikazati v obliki, ki je produkt normiranega števila od “-1” do “+1” (mantise) in najmanjšega mnogokratnika števila 2 (eksponenta), ki še dovoljuje zapis mantise v normirani obliki<sup>4</sup>, npr.

$$“-3,65” = “-0,9125” \cdot 2^2.$$

Na ta način način smo zahtevano operacijo prevedli v obliko:

$$“0,12” \cdot “-0,9125” \cdot 2^2.$$

Takšno množenje je sestavljeno iz dveh korakov:

1. Standardno množenje dveh normiranih števil.
  2. Dodatno množenje z mnogokratnikom binarne baze. To najlažje naredimo z že znano operacijo pomika (shift) rezultata prvega koraka v levo (v tem primeru za dve poziciji). Ta pomik lahko odštejemo od pomika, ki ga zahteva poravnavi pri množenju normiranih števil.

Glede na to, da gre pri tem za množenje s vrednostjo, ki je večja od "1", moramo pri 2. koraku s posebnim algoritmom oz. ukazom preveriti, če je prišlo do presežka.

#### **10.1.1.3 Deljenje**

Pri deljenju je potrebno upoštevati še nekatera dodatna pravila in nasveti.

V operacijah deljenja v zbirniku je format deljenca povečinoma večji od formata delitelja in rezultata. Npr. v 16-bitnih procesorjih je deljenec zapisan z 32 biti, deljenec pa s šestnajstimi. Rezultat deljenja je 16-bitno število, prav tako tudi ostanek (glej pogl. 4.2.2). To lahko zapišemo v naslednji obliki:  $32/16=16:16$ . Pri procesorjih, kjer računamo z 32-bitnimi podatki, potemtakem velja:  $64/32=32:32$ .

Očitno je torej pred deljenjem treba deljenec spraviti na višji obseg, ki nam bo omogočil izvesti normiranje. Pri 16-bitnem deljenju transformacijo njegovega formata iz 16 v 32 bitov najlažje dosežemo tako, da mu na levi strani dodamo še eno pomožno besedo (običajno sestavljeno iz ničel).

### *Primer:*

Deljenje "0,15" / "0,45" oz. 1333<sub>HEX</sub> / 3999<sub>HEX</sub>

naj bi dalo rezultat "0,333" oz.  $2aaa_{HEX}$ .

S povečanjem formata deljenca, ki ga zahteva operacija (npr. DIVS.W d0,d1 v MC68332) bomo izvršili operacijo:

$13330000_{\text{HEX}} / 3999_{\text{HEX}}$ ,

<sup>4</sup> Mantisa naj bo čim bližja maksimalni vrednosti, saj le tako ohranjamо največjo možno ločljivost.

rezultata pa bosta:

$5555_{\text{HEX}}$  (celoštevilski količnik) in  $1333_{\text{HEX}}$  (ostanek).

Količnik ne ustreza pričakovanemu rezultatu, temveč je dvakrat večji zato, ker smo z dodajanjem spodnje besede deljencu izvršili naslednjo operacijo:

$$[(“0,15” \cdot 2^{15}) \cdot 2^{16}] / (‘0,45’ \cdot 2^{15}) = “0,333” \cdot 2^{16} = (“0,333” \cdot 2) \cdot 2^{15},$$

kar je treba upoštevati pri končnem rezultatu<sup>5</sup>.

Najpogostejsi napaki, ki se pojavljata pri deljenju, sta presežek in deljenje z ničlo. Do prve pride takrat, ko delimo dejansko večje normirano število z manjšim (ne glede na povečanje formata). Rezultat bo takrat večji od “±1”, kar bo javil bit presežka OV (ali V) v statusnem registru. Če že obstaja potreba po takem deljenju, jo moramo izvesti z upoštevanjem eksponentov (podobno kot pri množenju).

Deljenje z ničlo je veliko nevarnejša napaka, ki ima lahko nepredvidljive posledice, saj je rezultat neskončno število. Procesorji običajno takrat samodejno generirajo ustrezen prekinitiv, ki jo programer izkoristi za sprožanje nadaljnijih ukrepov.

Deljenje je običajno časovno zelo dolgotrajna operacija (zlasti pri DSP-jih, kjer je lahko najmanj šestnajstkrat daljša od ostalih), zato se ji izogibamo ali nadomestimo z neko drugo operacijo.

Npr. namesto deljenja spremenljivke  $a$  s konstanto  $\text{konst}$ :

$$\frac{a}{\text{konst}}$$

raje vpeljemo množenje z novo konstanto, ki smo jo že vnaprej izračunali med inicializacijskim postopkom:

$$a \cdot \text{konst}_I, \quad \text{kjer je } \text{konst}_I = \text{konst}^{-1}.$$

Enako se splača pri izrazu

$$\frac{a}{b \cdot c}$$

izračunati produkt imenovalca in šele nato izračunati količnika, namesto dvakratnega deljenja z  $b$  in nato s  $c$ .

Pogostokrat imamo opravka s kombiniranimi operacijami množenja in deljenja enako normiranih spremenljivk, npr.

---

<sup>5</sup> **PAZI!** Povečanje formata z dodajanjem ničel ne prispeva k večji ločljivosti (simbolično):  $[(“0,15” \cdot 2^{15}) \cdot 2^{16}] \neq “0,15” \cdot 2^{31}!$

$$\frac{a \cdot b}{c}.$$

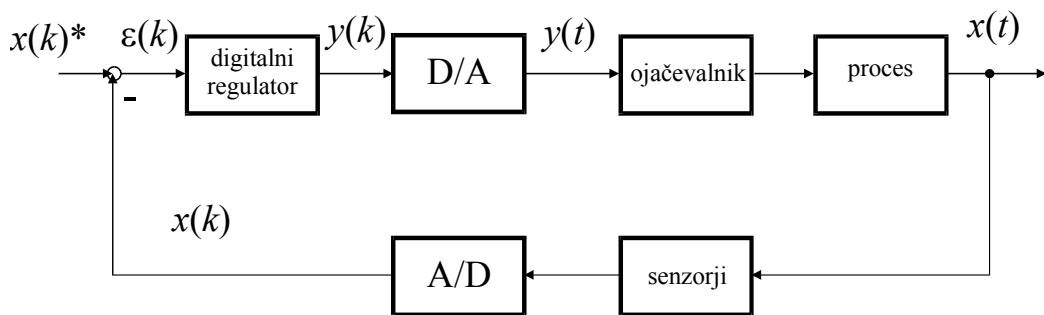
Sedaj že vemo, da imamo v primeru množenja  $(a \cdot b)$  opravka s 32-bitnim rezultatom. Tega ni potrebno zmanjšati na 16-bitno vrednost, saj bomo ta format uporabili takoj pri deljenju. Na koncu je eventualno potrebna le manjša poravnava v levo ali desno, odvisno od tipa spremenljivk.

## 10.2 Mikroračunalnik v regulacijskih sistemih

### 10.2.1 Kratka primerjava analognih in digitalnih regulatorjev

Klasični regulacijski sistemi so vrsto let uporabljali *analogne regulatorje*, ki so sestavljeni iz operacijskih ojačevalnikov, uporov in kondenzatorjev. Z različnimi kombinacijami teh je mogoče zgraditi vse elemente klasične regulacijske tehnike, kot so proporcionalni, diferencialni, in integralni člene ali njihove kombinacije, nizko- in visokopasovni filtri itd. Po definiciji sta frekvenčni pas in ločljivost signalov pri takšnih analognih vezjih visoka. Po drugi strani so glavne pomanjkljivosti analognih rešitev nefleksibilnost, visoka cena elementov, kot so množilniki, omejenost realizacije nekaterih funkcij, nezmožnost shranjevanja in numerične obdelave rezultatov itd.

Glavni sestavni del sodobnih *digitalnih regulatorjev* so mikroprocesorski elementi, kar omogoča veliko fleksibilnost (funkcijo regulatorja takoj spremenimo z drugačnim programom), bistveno manjšo občutljivost na motnje, izvajanje zapletenih algoritmov itd. Slika 10. 1 kaže osnovno blokovno shemo regulacijskega kroga z digitalnim regulatorjem



**Slika 10. 1: Regulacijski sistem z digitalnim regulatorjem**

Glavna pomanjkljivost digitalnih regulatorjev je v omejeni ločljivosti, kar v določenih primerih pripelje do resnih numeričnih problemov, ter omejen frekvenčni pas. Tabela 10.1 kaže primerjalne karakteristike obeh pristopov [1].

| Regulator        | Prednosti  | Pomanjkljivosti   |
|------------------|--|---|
| <i>Analogni</i>  | Široko frekvenčno področje<br>Visoka ločljivost<br>Enostavnost načrtovanja                                     | Staranje<br>Temperaturni drift<br>Fiksna (ozičena) struktura<br>Primernost le za manj zahtevne naloge |
| <i>Digitalni</i> | Programirljive rešitve<br>Neobčutljivost na motnje<br>Aplikacija sodobnih algoritmov<br>Možnost dodatnih nalog | Numerične težave<br>Običajno potreba po uporabi visoko zmogljivih procesorjev<br>Zapletena gradnja    |

**Tabela 10.1: Primerjava analognih in mikroprocesorskih (digitalnih) regulatorjev**

### 10.2.2 Problemi pri izbiri mikroprocesorskih regulatorjev

Večina digitalnih regulatorjev uporablja celoštevilsko aritmetiko, kjer je prikaz velikosti signala ali ojačenja možen le s končno ločljivostjo (npr. 8, 16 ali 32 bitov). Za prilagajanje vseh veličin temu področju je potrebno skaliranje oz. normiranje. Efekti končne dolžine besede (angl. finite wordlength) se kažejo v obliki *diskretizacijskega* ali *kvantizacijskega šuma* (angl. quantization noise), ki lahko privedejo celo do nestabilnosti celotnega sistema [1].

Druga posledica končne dolžine besede se kaže v *pogrešku zaokroževanja* (angl. roundoff error). Med zajemanjem in izdajanjem končnih vrednosti je običajno veliko aritmetičnih operacij. Zaradi končne dolžine besede vnese vsaka od njih določen pogrešek, ki se po vsakem ukazu kumulira. Npr. rezultat množenja v 32-bitnem procesorju je običajno shranjen v 64-bitni obliki (pri 16-bitnih pa  $16 \cdot 16 = 32$ ), ker pa ga moramo uporabiti tudi pri nadaljnjih operacijah, je potrebno skrajšanje na osnovni format 32 bitov. To pomeni, da smo izgubili spodnjih 32 bitov. Nadalje, ob povečanju vrednosti registra celoštevilskih procesorjev nad maksimalno vrednost pride do presežka, ko se spremeni predznak. Poseben problem pri normalizaciji je ravno preprečevanje presežka, saj moramo veličine normirati tako, da tudi pri največji vrednosti do tega ne pride. Če pri tem za nazivne vrednosti uporabimo le spodnji del besede (npr. 12 bitov pri formatu f4.12, glej pogl. 10.3.4.2), smo ločljivost še dodatno zmanjšali.

Zakaj se potem pri izbiri digitalnih regulatorjev za aplikacije v močnostni elektroniki ne odločamo za procesorje s plavajočo vejico, kjer ni zgoraj navedenih negativnih učinkov? Osnovni razlog je cena! Namen proizvajalcev je ponuditi procesorje, ki jih bomo lahko uporabili v številnih aplikacijah, kjer lahko cena procesorja odločilno vpliva na ceno končnega izdelka (npr. pralni stroji, klimatske naprave itd.). Multiprocesorski sistemi z vgrajenim matematičnim koprocesorjem so enostavno predragi in je njihova uporaba upravičena le pri reševanju zapletenih problemov.

## 10.3 Zgled zajemanja veličin, normiranja in realizacije nekaterih funkcij

Kot osnovo za ilustracijo aplikacije mikroprocesorskega sistema bomo vzeli vektorsko regulacijo pogona z asinhronskim motorjem (v nadaljevanju AM). V njej so obrazloženi

posamezni problemi, ki jih srečujemo v večini mikroračunalniških sistemov za regulacijo v realnem času:

1. Najkrajše (električne) časovne konstante konstante sistema so od nekaj ms do nekaj deset ms. Zadosti dobra izbira časa (intervala) vzorčenja je  $T_V \leq 1$  ms (glej tudi pogl. 10.3.3). Iz tega sledi zahteva po uporabi hitrega mikroprocesorja, ki bo lahko celotni regulacijski program izvršil znotraj intervala vzorčenja (npr. MC68332 ali DSP320F240). Zaradi konkurenčnosti končnega izdelka si ne smemo privoščiti "predobrega" in s tem tudi dragega mikroprocesorja (cene procesorjev za to aplikacijo so običajno do nekaj deset \$).
2. V programu so prisotni različni splošno uporabni algoritmi (npr. nizkopasovni filtri - členi prvega reda, regulatorji, trigonometrične funkcije, v nekaterih primerih pulzno-širinska modulacija napetosti itd.).
3. Mikroračunalniški sistem vsebuje različne vhodno/izhodne enote (A/D in D/A pretvornike, vezja za zajemanje pulzov iz inkrementalnega dajalnika, binarne vhode in izhode).
4. Procesni mikroračunalniški sistem mora imeti možnost komunikacije z nadrejenim računalnikom (običajno osebnim računalnikom prek serijskega protokola EIA-232).

V tem podoglavlju bomo obdelali naslednje teme:

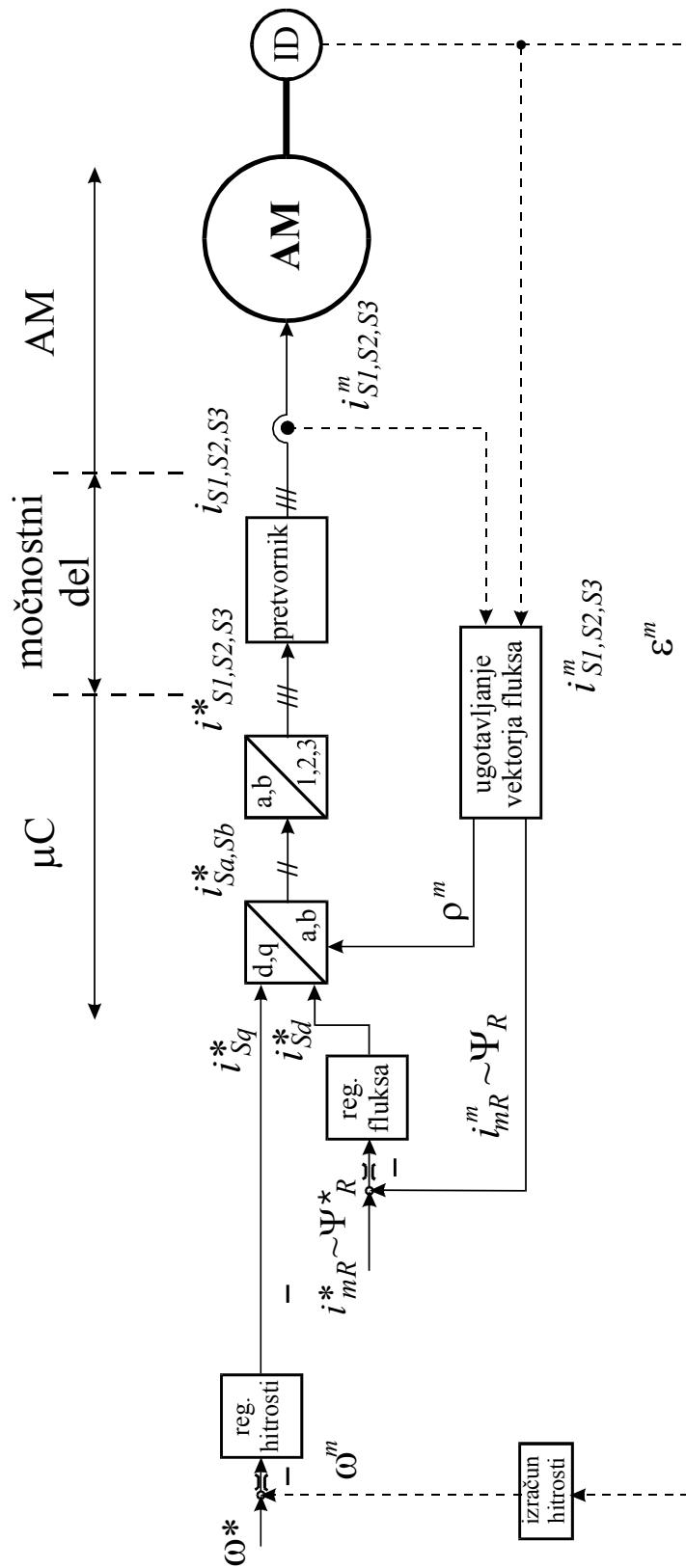
- Opis regulacijske naloge.
- Arhitektura mikroračunalnika.
- Struktura programa.
- Normiranje veličin in karakteristične matematične operacije.
- Realizacija nekaterih splošno uporabnih funkcij.

### **10.3.1 Opis regulacijske naloge**

Blokovno shemo regulacije hitrosti in/ali kota zasuka AM kaže slika 10. 2. Tukaj se ne bomo ukvarjali s podrobnim opisom fizikalnega ozadja celotnega problema, saj je to opisano v literaturi [npr. 2], temveč bomo obravnavali le posamezne splošno uporabne postopke in segmente algoritma. Videli bomo, da gre za dokaj zapleteno regulacijsko nalogo, ki jo mora mikroračunalnik izvajati sproti, v realnem času. Naj omenimo, da je to le eden izmed možnih pristopov k reševanju dane regulacijske naloge.

Pogon je sestavljen iz treh osnovnih delov (glej tudi sliko 10. 5):

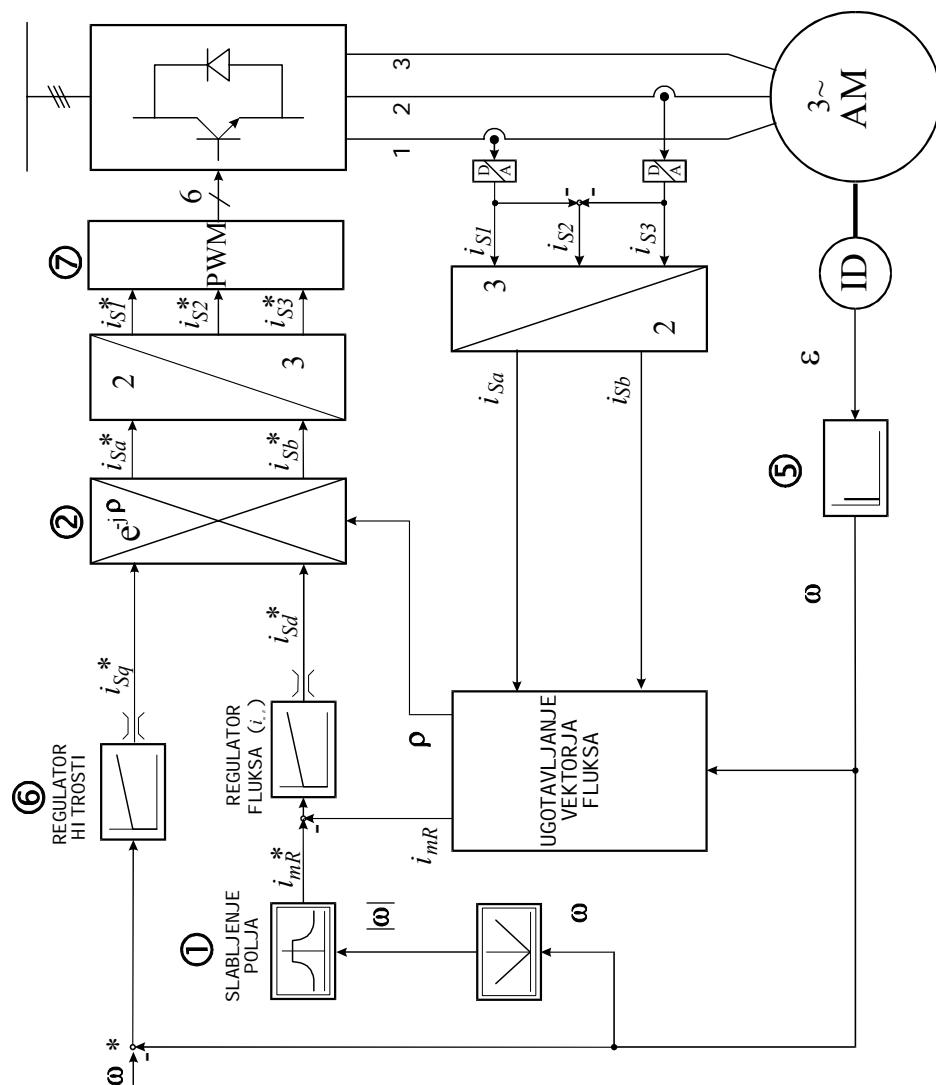
1. Reguliranec je **asinhronski motor**, običajno s kratkostično kletko (indukcijski motor).
2. **Procesni mikroračunalnik** je zadolžen za regulacijo toka ali napetosti, s katero napajamo motor. Pri tem je so potrebne meritve nekaterih veličin iz motorja (npr. napetosti, toka ter kota zasuka rotorja oz. njegovega odvoda - kotne hitrosti).
3. **Močnostni pretvornik** je v bistvu ojačevalnik, ki izhodne signale pretvornika pretvorí v dejanske napetosti ali tokove na statorskem navitju motorja. V našem primeru imamo opravka s tokovnim virom, kjer mikroračunalnik narekuje obliko toka. V praksi je pogosto za to zadolžena t.i. pulzna-širinsko modulacija (angl. Pulse Width Modulation - PWM), kjer računalnik generira pulze za proženje tranzitorjev v pretvorniku po določenem algoritmu (pogl. 10.3.5.7).



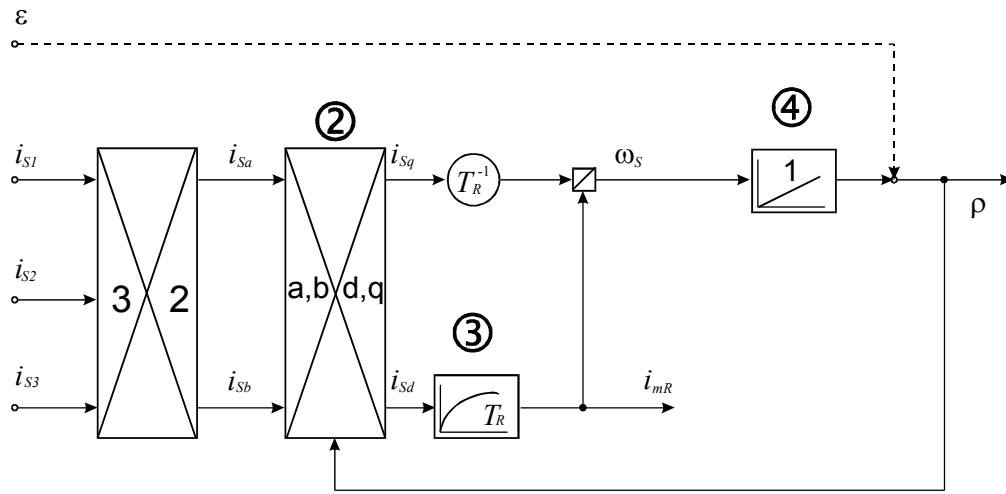
Slika 10. 2: Načelna blokovna shema vektorske regulacije asinhronskega motorja

Kratek opis naloge:

Mikroračunalnik primerja želeno vrednost vrtilne hitrosti ( $\omega^*$ ) z dejansko ( $\omega''$ ). Le-to dobimo z odvajanjem kota rotorja, ki ga merimo s pomočjo inkrementalnega dajalnika (ID - glej tudi pogl. 12.5.2), po času. Razlika hitrosti je vhodna veličina v regulator (običajno PI), izhodna veličina pa tok  $i_{Sd}^*$ , ki je proporcionalen električnemu momentu. Za regulacijo fluksa motorja je zadolžen drugi regulator (prav tako PI), na katerega pripeljemo razliko med želenim in dejanskim fluksom. Slednjega ne moremo neposredno izmeriti, lahko pa ga izračunamo iz enačb v bloku "ugotavljanje vektorja fluksa". Za to potrebujemo informacijo o dejanskih statorskih tokih in trenutni poziciji rotorja. Izhodno veličino iz regulatorja fluksa  $i_{Sd}^*$  skupaj z  $i_{Sq}^*$  pripeljemo na transformacijski blok (množenje in trigonometrične funkcije), katerega rezultat sta  $a$  in  $b$  komponenta vektorja statorskega toka v fiksni koordinatnem sistemu. Po še eni transformaciji (množenje in seštevanje) dobimo tri želene statorske toke  $i_{S1}^*$ ,  $i_{S2}^*$  in  $i_{S3}^*$ . To so hkrati vhodne vrednosti v tokovni močnostni pretvornik. Funkcije, ki jih mora izvajati mikroračunalnik, bodo jasnejše, če si ogledamo podrobnejši regulacijski shemi na slikah 10. 3 in 10. 4.



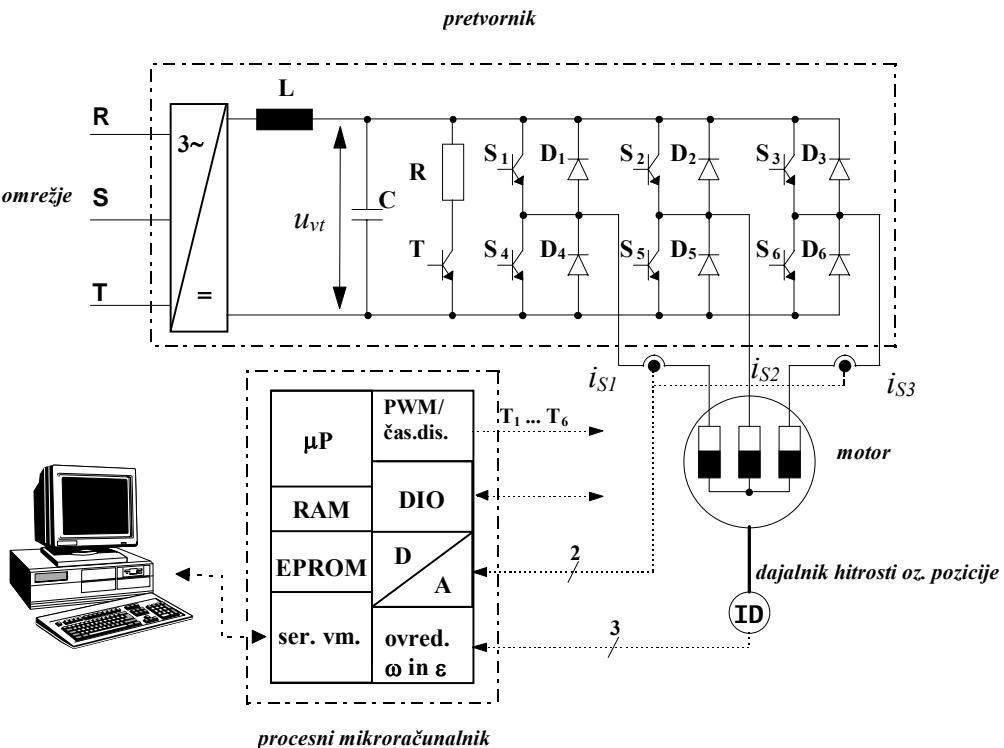
Slika 10. 3: Podrobnejša blokovna shema regulacije AM



Slika 10. 4: Podrobna blokovna shema bloka "ugotavljanje vektorja fluksa" s slike 10. 3 za dvopolni stroj

### 10.3.2 Arhitektura mikroračunalnika

Slika 10. 5 kaže načelno zgradbo mikroračunalnika za izvajanje regulacije asinhronskega motorja.



Slika 10. 5: Karakteristična konfiguracija modernega pogona z asinhronskim motorjem

Osnovne komponente mikroračunalnika so:

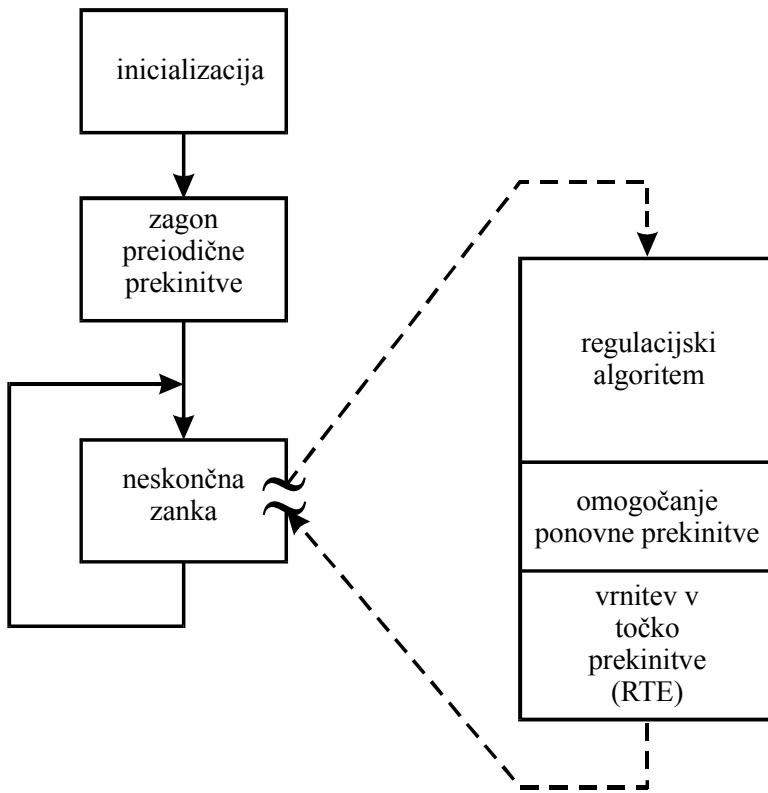
1. **Mikroprocesor ( $\mu$ P).**
2. **Pomnilnik** je potreben za shranjevanje regulacijskega programa in tabel (običajno bralni pomnilnik, npr. EPROM ali Flash EEPROM) ter spremenljivk (bralno-pisalni pomnilnik - RAM).
3. **Komunikacijska enota** (običajno serijski vmesnik EIA-232) skrbi za povezovanje z nadrejenim mikrorračunalnikom (npr. osebni računalnik - PC), katerega glavna naloga je določanje načina delovanja procesnega računalnika, parametrov in želenih vrednosti, kakor tudi prikaz tekočih spremenljivk. V primeru kompleksnejših pogonov je naloga PC-ja hierarhično organiziranje in koordiniranje več procesnih računalnikov.
4. **Digitalne vhodno-izhodne enote (DIO)** so namenjene zajemanju (npr. signalov iz inkrementalnega dajalnika, krmilnih pulzov, posebne tipkovnice) ali oddajanju binarnih signalov (npr. krmiljenje prikazovalnika s tekočimi kristali, proženje močnostnih stikalnih elementov).
5. **PWM enota** je nujna za generiranje prožilnih signalov za tranzistorje, če želimo modulirati izhodno napetost želene oblike. Pri starejših izvedbah je za to zadolžen poseben čip, nekateri novejši mikrokontrolerji (npr. MC68332 ali TMS320F240) pa že vsebujejo lastno PWM enoto.
6. **Analogne vhodno-izhodne enote.** Zajemanje analognih procesnih veličin (kot so npr. toki ali signali iz analognih dajalnikov pozicije), morebitno filtriranje in pretvorba v digitalno obliko, ki je primerna za obdelavo v  $\mu$ C, je naloga analogno-digitalnih (A/D) pretvornikov. V ta namen se pri zahtevnejših sistemih ponavadi uporablajo 10- (ločljivost  $2^{10} = 1024$ ) ali 12-bitni (ločljivost  $2^{12} = 4096$ ) A/D pretvorniki s kratkim časom pretvorbe (nekaj  $\mu$ s). Z ločljivostjo ne gre pretiravati, saj je njena zgornja meja odvisna od točnostnega razreda meritnikov ter velikosti motenj, ki so zelo pogost pojav pri takšnih sistemih. Zlasti pri zajemanju signalov iz meritnika hitrosti lahko uporabimo manj zmogljive 8-bitne pretvornike. Dodaten razlog za pravilno oceno potrebne zahtevnosti pretvornikov je cena, ki z višanjem zmogljivosti strmo narašča. Digitalno-analogni (D/A) pretvorniki imajo dvojno funkcijo:
  1. Pri časovno-diskretnem pretvorniku je njihova naloga pretvorba želenih vrednosti tokov, izračunanih v računalniku, v analogne veličine.
  2. Regulacijski algoritem, ki se izvaja v  $\mu$ C, vsebuje veliko vmesnih spremenljivk, ki jih je zelo težko sproti spremamljati (npr. vsakih 500  $\mu$ s). Njihove trenutne vrednosti najlaže opazujemo tako, da jih preko D/A pretvornika prikažemo na osciloskopu.

### 10.3.3 Struktura programa

Računalnik rabi nek končni čas za izvajanje regulacijskega programa. Na začetku enega programskega cikla se izmenjajo podatki med periferijo (procesom) in mikrorračunalnikom. Zaradi zakonitosti, ki veljajo za diskrete sisteme, je potrebno ta čas (interval ali čas vzorčenja  $T_V$ ) med dvema zajemanjima vhodnih ali izdajama izhodnih signalov vnaprej fiksno določiti. V praksi ponavadi izberemo čas, ki je vsaj šest do desetkrat kraši od najmanjše časovne konstante sistema<sup>6</sup>.

---

<sup>6</sup> Za razliko od strogih zahtev diskretnih regulacijskih sistemov je pri obdelavi signalov frekvence vzorčenja vsaj dvakrat večja od najvišje frekvence v sistemu.

**Slika 10. 6: Principialna struktura programa**

Načeloma je vsak program za regulacijo v realnem času sestavljen iz naslednjih delov (slika 10. 6):

1. **InicIALIZACIJA** je del programa, ki se ponavadi izvrši le enkrat. Osnovni namen inicializacije je postavitev začetnih vrednosti spremenljivk ali ugotavljanje začetnega stanja procesa (npr. postopek identifikacije parametrov motorja, ki se izvaja pred zagonom).
2. **Zagon periodične prekinitve**. Od tega trenutka naprej bo tekoči program (glej naslednjo točko) periodično prekinjena s prekinitvenim podprogramom. Perioda prekinitve je enaka času vzorčenja  $T_V$ .
3. Po zagonu periodične prekinitve preide izvajanje programa v **neskončno zanko**, ki jo bo ciklično prekinjal prekinitveni podprogram. Znotraj neskončne zanke lahko izvajamo časovno nekritične funkcije, kot je npr. komunikacija z nadrejenim računalnikom oz. uporabnikom, ki lahko procesorju posreduje nove informacije (npr. novo želeno hitrost motorja). Zavedati se moramo, da je programu v zanki na voljo le čas od vrnitve iz prekinitvenega podprograma do naslednje prekinitve, kar pomeni, da ne moremo zagotoviti izvajanja celotnega algoritma v zanki do nastopa naslednje prekinitve. Zato lahko v zanko postavimo le algoritme, ki jih lahko prekinjamo brez posledic.
4. **Glavni regulacijski program** se nahaja v prekinitvenem podprogramu. S tako izbiro delovanja smo omogočili izvajanje začetka regulacijskega programa vsakih  $T_V$ . Pri tem pa moramo paziti na to, da s tem nismo zagotovili tudi vedno enakega trajanja samega podprograma; to je odvisno od notranjih pogojnih skokov oz. zank in se lahko med

obratovanjem spremenja<sup>7</sup>. Pomembno je le, da je maksimalni čas trajanja regulacijskega algoritma vedno krajši od periode prekinitve, ker bi v nasprotnem primeru lahko prišlo do prekinjanja samega prekinitvenega podprograma, kar bi imelo nepredvidljive posledice<sup>8</sup>.

#### 10.3.4 Normiranje veličin in karakteristične matematične operacije

##### 10.3.4.1 Izračun osnovnih vrednosti

V poglavju 10.1 smo že povedali, da je prvi korak pri normiraju izbira referenčne delovne točke, v kateri so znane vrednosti vseh fizikalnih veličin procesa (npr. nazivno obratovanje ali kratek stik pri motorjih). V našem primeru bomo upoštevali nazivne vrednosti, saj jih je večina dostopna na napisni tablici ali katalogih, ostale pa lahko izračunamo iz obstoječih podatkov. V tabeli (10. 11) so podatki v koherentnih enotah za motor in pretvornik (napajalnik).

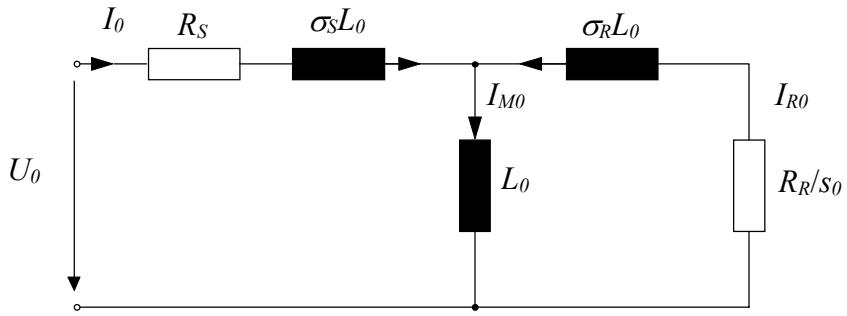
| Vezava   |                           | zvezda   |
|--|---------------------------|--|
| Nazivna moč  | $P_0$                     | 1,2 kW   |
| Nazivna fazna napetost                             | $U_0$                     | 220 V  |
| Nazivna frekvenca<br>(sinhronska vrtilna hitrost)  | $f_{10}$<br>( $n_S$ )     | 50 Hz<br>(3000 min <sup>-1</sup> )                 |
| Nazivna kotna hitrost<br>(nazivna vrtilna hitrost) | $\omega_0$<br>( $n_0$ )   | 148,7 s <sup>-1</sup><br>(1420 min <sup>-1</sup> ) |
| Nazivni fazni tok                                  | $I_0$                     | 3,4 A (pri 380 V,<br>50 Hz)                        |
| Število polovih parov                              | $p$                       | 2  |
| Faktor moči  | $\cos \phi$               | 0,86   |
| Statorska upornost                                 | $R_S$                     | 3,839 Ω  |
| Rotorska upornost                                  | $R_R$                     | 2,980 Ω  |
| Medsebojna induktivnost                            | $L_0$                     | 0,2587 H   |
| Statorska induktivnost                             | $L_S = L_0(1 + \sigma_S)$ | 0,26627 H  |
| Rotorska induktivnost                              | $L_R = L_0(1 + \sigma_R)$ | 0,26611 H  |
| Maksimalni tok pretvornika                         | $I_{max}$                 | 25 A   |

Tabela 10.2: Nazivne vrednosti AM

V nazivni delovni točki, v stacionarnem stanju, lahko dogajanja v AM opišemo z enofazno nadomestno shemo na sliki 10. 7.

<sup>7</sup> Zaradi tega se načeloma zajemanje vhodnih signalov ter izdajanje izhodnih signalov iz prejšnjega cikla izvaja vedno na začetku prekinitvenega podprograma.

<sup>8</sup> Npr. če je  $T_V = 1$  ms, bo maksimalni čas izvajanja regulacijskega algoritma krajši (npr. 850 μs). Preostalih 150 μs se bo izvajal program v neskončni zanki.



**Slika 10. 7: Nadomestna enofazna shema AM v stacionarnem stanju**

Najprej bomo normirali osnovne veličine, iz katerih sledijo tudi norme ostalih veličin. V našem primeru bomo izbrali fazno napetost, fazni tok ter kotno frekvenco. Te vrednosti imenujemo *osnovne vrednosti* (angl. base values) [30]. Osnovne vrednosti so enake maksimalni trenutni vrednosti fizikalne veličine v nazivni delovni točki. V tablici so podane efektivne vrednosti toka in napetosti, torej velja:

$$I_B = \sqrt{2}I_0 = \sqrt{2} \cdot 3,4 \text{ A} = 4,8 \text{ A},$$

$$U_B = \sqrt{2}U_0 = \sqrt{2} \cdot 220 \text{ V} = 311,13 \text{ V}$$

ter

$$\omega_B = 2\pi f_{l0} = 2\pi f_{l0} = 314,159 \text{ rad s}^{-1}.$$

Z izbiro osnovnih, referenčnih vrednosti zgornjih veličin bomo njihove trenutne vrednosti odslej označevali z relativnimi (angl. pu - per unit: na enoto) vrednostmi, npr. za tok:

$$i_S^{rel} = \frac{i_S}{I_B},$$

kjer je  $i_S$  trenutna vrednost statorskoga toka. Tako bo relativni tok v trenutku, ko je vrednost sinusnega toka enaka negativni amplitudi nazivnega toka, enak "-1" (glej tudi pogl. 10.1), pri maksimalni možni vrednosti toka, ki jo dovoljuje pretvornik (glej tudi tabelo (10. 11)) pa

$$i_S^{rel} = \frac{i_{max}}{I_B} = \frac{25 \text{ A}}{4,8 \text{ A}} = "5,2083".$$

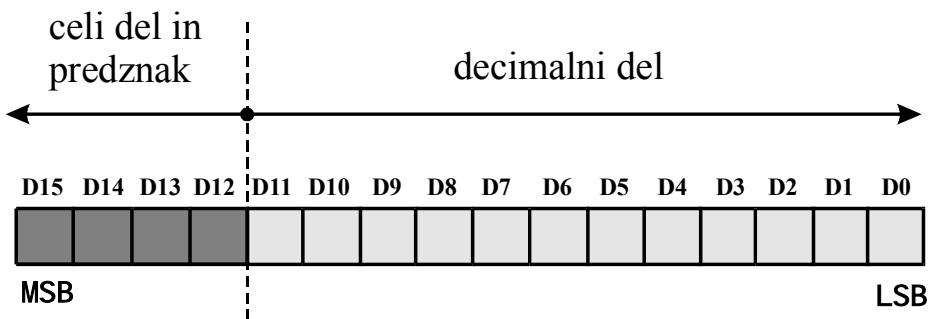
#### 10.3.4.2 Izbira formata zapisa digitalnih vrednosti

Sedaj moramo izbrati še šestnajstiški ekvivalent (*format*) osnovnim normiranim vrednostim, ki smo jih v prejšnjih poglavjih označili z "1". Osnovno besedo razdelimo na, pogojno rečeno, *celi* in *decimalni* del. Pri tem moramo upoštevati ločljivosti osnovne besede (v našem primeru  $\pm 2^{15}$ ) in hkrati paziti, da maksimalna vrednost ne bo presegla številskega področja. V primeru formatiranja toka vidimo, da je maksimalna vrednost 5,2083-krat večja od osnovne.

Za zapis celega števila 5 potrebujemo vsaj tri bite ( $2^3 - 1 = 7$ ), glede na to, da je tok lahko pozitiven ali negativen, pa moramo dodati še en bit. Vrednosti “-7” ustreza  $8_{\text{HEX}} = 1000_{\text{BIN}}$ , vrednosti “+7” pa  $7_{\text{HEX}} = 0111_{\text{BIN}}$ . Za decimalni del (“0” - “1”) potem takem preostaja še 12 bitov, kar pomeni, da je njegova ločljivost

$$\frac{1}{2^{12}} = 0,0002414.$$

Takšen format simbolično zapišemo kot f4.12 (slika 10. 8)



*Slika 10. 8: Grafična ponazoritev formata f4.12*

Pretvorba iz relativne normirane vrednosti v formatu f4.12 poteka torej v dveh delih:

1. Celi del pretvorimo v 4-bitni dvojiški komplement.
2. Decimalni del dobimo tako, da najprej izračunamo desetiški ekvivalent decimalnega dela ( $x$ ), ki ga nato pretvorimo v šestnajstistiško oz. binarno obliko (“ $x$ ” je decimalni del normirane vrednosti):

$$x = "x" \cdot 2^{12} (\text{DEC}) = "x" \cdot 4096 (\text{DEC}) \rightarrow \text{HEX (BIN)}. \quad (10. 2)$$

Oglejmo si nekaj primerov digitalnih ekvivalentov toka:

| Dejanska vrednost (A) | Normirana vrednost | Digitalna vrednost (HEX) |
|-----------------------|--------------------|--------------------------|
| -4,8                  | “-1”               | f000                     |
| +4,8                  | “+1”               | 1000                     |
| +25                   | “5,2083”           | 5355                     |
| +2,4                  | “0,5”              | 0800                     |

#### 10.3.4.3 Aritmetične operacije med digitalnimi formatiranimi števili

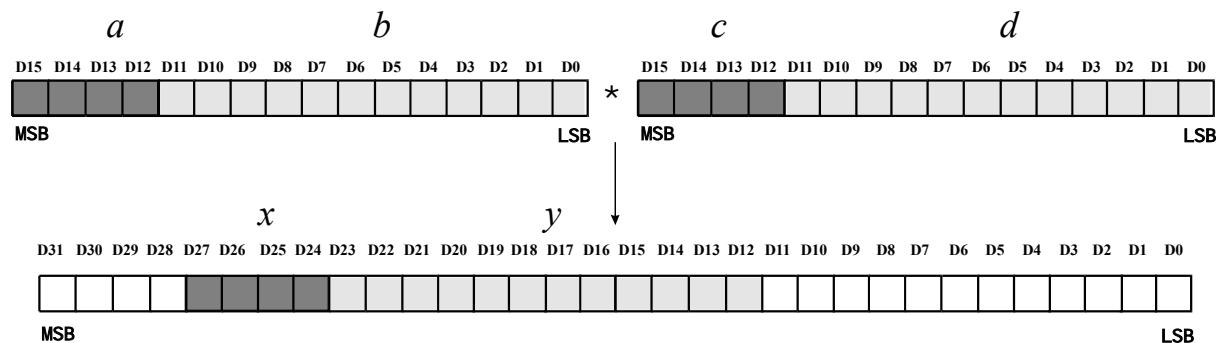
Preverimo, ali logika dvojiškega komplementa še vedno drži: s seštevanjem dveh tokov 25 A in -4,8 A dobimo 20,2 A, seštevanje njunih digitalnih ekvivalentov pa da

$$\begin{array}{r}
 5355_{\text{HEX}} \\
 + f000_{\text{HEX}} \\
 \hline
 14355_{\text{HEX}}
 \end{array}$$

prenos, nepomembno

Če dobljeno število pretvorimo nazaj v relativno vrednost po postopku, ki je inverzen opisanemu v (10. 2), dobimo "4,2083". Decimalni del smo izračunali tako, da smo  $355_{\text{HEX}} = 853_{\text{DEC}}$  delili s  $4096_{\text{DEC}}$  ter dobili  $0,2083_{\text{DEC}}$ . Dejanska vrednost toka pa je " $4,2083 \cdot 4,8 \text{ A} = 20,19984 \text{ A}$ , torej izračun drži!"

Pri množenju in zlasti pri deljenju je treba biti nekoliko bolj pozoren, saj postopek navadnega množenja, ki smo ga opisali v poglavju 10.1.1.2, ne drži več<sup>9</sup>. Naš cilj je zmnožiti dve števili  $a \cdot b$  in  $c \cdot d$  ( $a$  in  $c$  sta cela dela,  $b$  in  $d$  pa decimalna dela množenca in množitelja) v formatu f4.12 in dobiti rezultat  $x.y$  v enakem formatu. Slika 10. 9 nazorno kaže format zmnožka.



*Slika 10. 9*

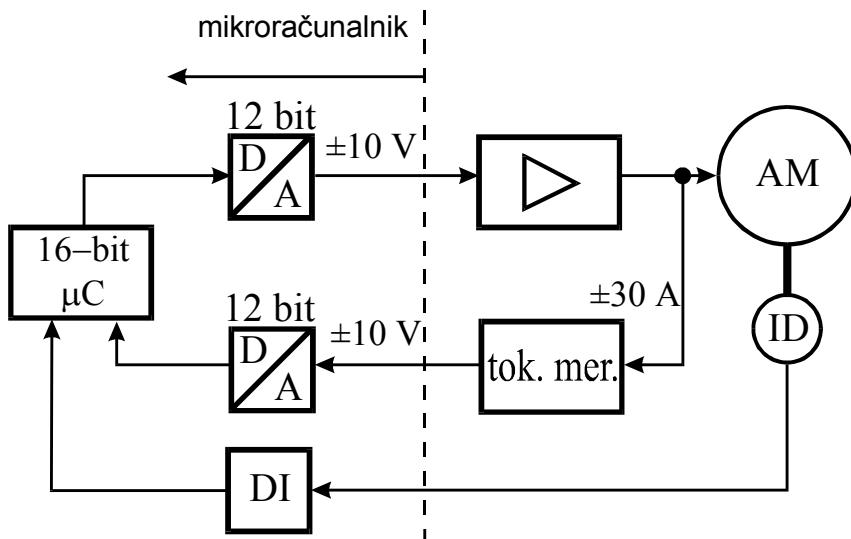
Rezultat v f4.12 bo premaknjen v desno od MSB za 4 bite oz. levo od LSB za 12 bitov. Če hočemo dobiti rezultat v želenem formatu, moramo ustreznno poravnati njegovo vsebino ter upoštevati le zgornjo ali spodnjo besedo tako prilagojenega rezultata. Po premisleku, vidimo, da smo na enak problem naleteli v pogl. 10.1.1.2, kjer smo ob operaciji  $f1.15 \cdot f1.15$  dobili rezultat v obliki f2.30. Od tod tudi potreba po premiku za 15 bitov v desno in upoštevanju le LSW.

Naj tukaj opozorimo še na potrebno previdnost pri vseh operacijah med veličinami, ki niso formatirane na enak način. To je sicer dovoljeno, pred vsako operacijo pa moramo skrbno premisliti kako bomo povezali različna formata.

#### 10.3.4.4 Prilagajanje vhodnih in izhodnih vrednosti normam

Nekatere izmed normiranih veličin v mikroprocesorju smo zajeli s pomočjo merilnikov ali pa jih bomo morali posredovati procesu, kjer imamo opravka z dejanskimi fizikalnimi veličinami. Tukaj bomo pokazali le primera zajemanja preko A/D pretvornika (toka) ter diskretnega dajalnika pulzov (kot rotorja), kar kaže slika 10. 10.

<sup>9</sup> Sedaj lahko rečemo, da smo imali tam opravka z množenjem števil v formatu f1.15!



**Slika 10. 10: Osnovna aparatura oprema za regulacijo asinhronskega motorja**

#### 10.3.4.4.1 Zajemanje preko A/D in D/A pretvornika

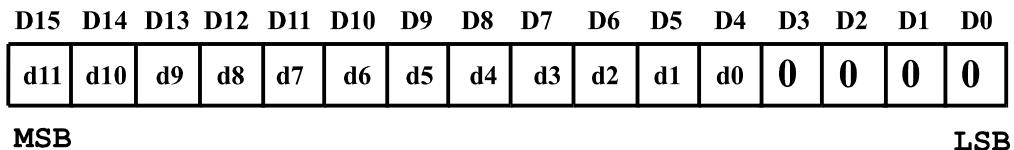
V konkretnem primeru zajemamo tok s tokovnim merilnikom. Vhodnemu območju  $\pm 30 \text{ A}$  ustreza izhodna napetost  $\pm 10 \text{ V}$ . To napetostno območje je značilno za A/D pretvornike, zato lahko izhod merilnika priključimo neposredno na pretvornik (ob dodatnih zaščitnih ukrepih, npr. proti prenapetosti). Ločljivost pretvornika naj bo 12-bitna (4096 enot), torej je ločljivost zajemanja napetosti  $20 \text{ V}/4096 = 4,88 \text{ mV}$ , ločljivost merjenega toka pa  $60 \text{ A}/4096 = 14,65 \text{ mA}$ . Nekatere karakteristične številske vrednosti A/D pretvornika in ustrezne toke kaže tabela 10.3. Pri tem predpostavljamo, da imamo opravka z bipolarnim pretvornikom, torej takim, ki upošteva logiko dvojiškega komplementa (glej tudi pogl. 12.4).

| Merjeni tok<br>(A) | Napetost na A/D<br>(V) | HEX 12-bit | HEX 16-bit<br>(leva poravnava) |
|--------------------|------------------------|------------|--------------------------------|
| -30                | -10                    | 800        | 8000                           |
| -15                | -5                     | c00        | c000                           |
| -0.01465           | -0.00488               | fff        | fff0                           |
| 0                  | 0                      | 000        | 0000                           |
| 0.01465            | 0.00488                | 001        | 0010                           |
| 15                 | 5                      | 3ff        | 3ff0                           |
| 30                 | 10                     | 7ff        | 7ff0                           |

**Tabela 10.3: Relacije med merjenimi toki, vhodnimi napetostmi in digitalnimi vrednostmi 12-bitnega pretvornika**

Z ozirom na to, da se format pretvornika (12-bitov) ne ujema s formatom mikroračunalnika (16 bitov), je potrebna ustrezna prilagoditev. Če hočemo, da bo logika dvojiškega komplementa veljala pri obeh formatih, moramo bite pretvornika “poravnati v levo” (angl. left justify) znotraj mikroračunalniške besede (tabela 10.3). V tem primeru se bosta MSB

obeh formatov ujemala, prazne lokacije pa bodo zapolnjene z ničlami (slika 10. 11). Poravnava lahko izvršimo softversko (pomik za štiri bite v levo) ali hardversko (z ustrezno povezavo podatkovnih bitov pretvornika in podatkovnega vodila procesorja).



*Slika 10. 11: Leva poravnava podatkovnih bitov 12-bitnega pretvornika (d11-d0) v 16-bitni besedi mikroračunalnika (D15-D0)*

Naslednja naloga bo prilagajanje digitalnih ekvivalentov merjenega toka normiranim vrednostim, ki jih poznamo iz poglavja 10.3.4.2. Normirani digitalni ekvivalent toku 30 A je

$$\frac{30\text{A}}{4,8\text{A}} \cdot 1000_{\text{HEX}} = 6400_{\text{HEX}}.$$

Če to primerjamo s poravnano vrednostjo A/D pretvornika, ki ustreza toku 30 A ( $7ff0_{\text{HEX}}$ ), lahko izračunamo faktor s katerim je treba množiti vhodno vrednost toka za prilagoditev formatu f4.12:

$$\frac{6400_{\text{HEX}}}{7ff0_{\text{HEX}}} = \frac{25600_{\text{DEC}}}{32752_{\text{DEC}}} = "0,78163" \rightarrow 0c82_{\text{HEX}}.$$

Poglejmo, ali izračun zares drži? Pri nazivnem toku 4,8 A bomo na A/D pretvorniku dobili vrednost

$$4,8\text{A} \cdot \frac{7ff_{\text{HEX}}}{30\text{A}} = 4,8\text{A} \cdot \frac{2047_{\text{DEC}}}{30\text{A}} = 327,5_{\text{DEC}} = 148_{\text{HEX}}$$

oz. po poravnavi v levo  $1480_{\text{HEX}}$ . Po množenju z  $0c82_{\text{HEX}}$  v mikroprocesorju dobimo:

$$1480_{\text{HEX}} \cdot 0c82_{\text{HEX}} = 0100\ 6900_{\text{HEX}},$$

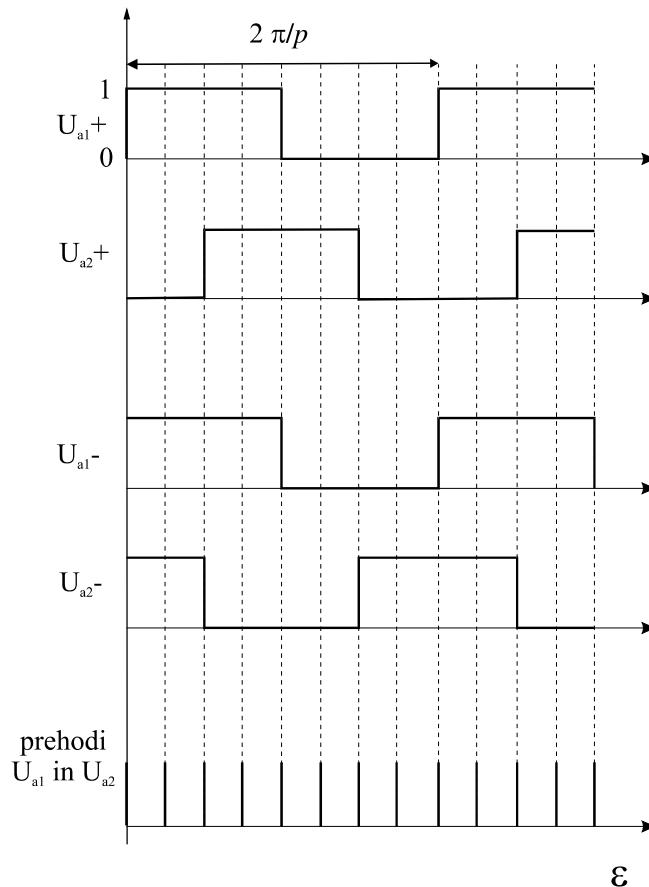
po poravnavi<sup>10</sup> v format f4.12 pa  $0100_{\text{HEX}}$ , kar je pričakovani rezultat.

Na podoben problem naletimo pri pošiljanju vrednosti na D/A pretvornik. Postopek je obraten, pri tem pa je treba paziti na format izhodnega podatka, ločljivost pretvornika in zmogljivost močnostnega ojačevalnika.

#### 10.3.4.4.2 Zajemanje kota prek inkrementalnega dajalnika

<sup>10</sup> Štiri bite v levo - upoštevaj MSW ali dvanaest bitov v desno - upoštevaj LSW.

Drugačnega zajemanje in prilagajanje vhodne veličine srečamo pri merjenju kota rotorja. Največkrat izvajamo takšno meritev z inkrementalnim dajalnikom (glej tudi pogl. 12.5.2), ki ob vrtenju generira dva vlaka za  $90^\circ$  premaknjenih pulzov ( $U_{a1}$  in  $U_{a2}$  na sliki 10. 12). Trenutni seštevek vseh prehodov obeh vlakov pulzov na digitalnih vhodih (DI na sliki 10. 10) pomeni informacijo o trenutnem kotu.



**Slika 10. 12: Oblike pravokotnih signalov iz inkrementalnega dajalnika z ozirom na smer vrtenja (+ ali -)**

Predpostavimo, da imamo inkrementalni dajalnik z 2048 pulzi na vrtljaj. Seštevek vseh prehodov obeh vlakov pulzov na en vrtljaj ( $2\pi$ ) bo štirikrat tolikšen:  $8192_{DEC} = 2000_{HEX}$ .

Za štetje so običajno zadolžene posebne enote, ki so sposobne šteti pulze navzgor ali navzdol, odvisno od smeri vrtenja. Štetje je sprotno in neodvisno od programa v CPU-ju, kar omogoča zajemanje zelo hitro spreminjačega se kota. Enote za zajemanje signalov z dajalnika so bodisi del mikrokontrolerja (npr. TPU pri MC68332 in QEP pri TMS320F240) ali pa ločena vezja (npr. LS2000). Ločljivost števcev je običajno 16 bitov, ponekod pa celo več.

Kot motorja rabimo pri nadaljnjih izračunih v dveh osnovnih blokih (slika 10. 3) za:

1. ugotavljanje hitrosti (odvod kota), kjer rabimo razliko kota med dvema vzorčenjima (glej tudi pogl. 10.3.5.5) ter za

2. transformacijo v koordinatni sistem rotorskega fluksa (blok “ugotavljanje vrednosti fluksa”), kjer moramo izračunati sinus in kosinus kota.

Sinus in kosinus sta periodični funkciji, kjer dobimo enako vrednost za kot  $\varepsilon$  v intervalu  $[0, 2\pi]$ , kakor tudi za kot  $\varepsilon + n \cdot 2\pi$ , kjer je  $n$  celo število. Ker sta obe trigonometrični funkciji podani tabelarično (pogl. 10.3.5.2.3), je treba izmerjeni kot (to je trenutni odčitek števca) limitirati na periodo (t.i. *modul*  $2\pi$ ). To dosežemo z enostavnim trikom:

V našem primeru ustreza periodi vrednost  $2000_{\text{HEX}}$ . Zato bomo vrednost števca “maskirali” na to območje, npr. z ukazom **AND I.W #\\$2000,d0**, če je podatek shranjen v registru **d0**. Na ta način upoštevamo le del vrednosti števca, ki je pod  $2\pi$ , vse kar je nad to vrednostjo (mnogokratnik  $2\pi$ ), pa nima pomena. Seveda se problem nekoliko zaplete, če ločljivost dajalnika ni enaka potenci števila dve.

Dobljeni rezultat je načeloma treba še normirati tako, da bo maksimalna vrednost kota  $2\pi$  ustreza lokaciji zadnjega podatka v tabeli trigonometrične funkcije (glej tudi pogl. 10.3.5.2.3).

#### 10.3.4.5 Normiranje parametrov

Po normirjanju osnovnih veličin (napetost, hitrost, tok pri motorju) sledi še normiranje ostalih parametrov in elektromagnetičnih veličin (npr. fluksa, pogl. 10.3.5.1). V tem podoglavlju bomo pokazali princip normiranja samo dveh parametrov motorja: upornosti in induktivnosti. Pri tem izhajamo iz nadomestne enofazne sheme motorja v stacionarnem stanju na sliki 10. 7.

#### **Normiranje upornosti:**

Ob osnovni vrednosti toka bo padec napetosti na statorski upornosti:

$$U_{SR} = I_B R_S = 4,8 \cdot 3,84 \text{ V} = 18,432 \text{ V},$$

kar normirano na osnovno vrednost napetosti znaša  $18,432/311,3 = “0,0592”$ .

Iz tega sledi, da je normirana upornost tudi “0,0592”, digitalni ekvivalent zapisan v formatu f4.12 pa:  $0,0592 \cdot 4096_{\text{DEC}} = 242_{\text{DEC}} = f2_{\text{HEX}}$ .

#### **Normiranje stresane induktivnosti:**

Nazivni padec napetosti na stresani induktivnosti  $\sigma_S L_0$  ( $L_S - L_0 = 0,00757 \text{ H}$ , tabela 10.2), to je ob nazivnem toku in frekvenci  $\omega_{f0} = 2\pi f_{10}$  je določen z izrazom:

$$U_{S\sigma L} = I_B \cdot 2\pi f_{10} \sigma_S L_0 = 4,8 \cdot 314,159 \cdot 0,00757 \text{ V} = 11,42 \text{ V},$$

oz. normirano “0,037”. Glede na to, da smo pri izračunu upoštevali normirani tok in normirano hitrost (“1”), ustreza izračunana vrednost že stresani induktivnosti. Njen digitalni ekvivalent v formatu f4.12 je  $0,037 \cdot 4096_{\text{DEC}} = 152_{\text{DEC}} = 98_{\text{HEX}}$ .

Poglejmo sedaj, kako bi potekal izračun nazivnega padca napetosti na stresani induktivnosti v računalniku:

1. Množenje  $\omega_{I0}$  in  $\sigma_S L_0$  (oba v f4.12):  $1000_{\text{HEX}} \cdot 98_{\text{HEX}} = 98000_{\text{HEX}}$ ,
2. Prilagajanje rezultata množenja<sup>11</sup> formatu f4.12:  $98_{\text{HEX}}$ ,
3. Množenje  $\omega_{I0} \cdot \sigma_S L_0$  z  $I_B$ :  $98_{\text{HEX}} \cdot 1000_{\text{HEX}} = 98000_{\text{HEX}}$ ,
4. Poravnavo rezultata (padca napetosti) v f4.12:  $98_{\text{HEX}}$ .

#### 10.3.4.6 Alternativni način normiranja in formatiranja

Vse veličine normiramo glede na maksimalne vrednosti (npr. maksimalni tok pretvornika ali motorja, napetost pretvornika in hitrost motorja). Njim dodelimo maksimalne številske vrednosti, ki jih lahko zapišemo s šestnajstimi biti:  $8000_{\text{HEX}}$  oz  $7fff_{\text{HEX}}$ . Takšen zapis bi v bistvu ustrezal formatu f1.15. V naslednjem koraku izračunamo digitalne ekvivalente nazivnih vrednosti osnovnih in izvedenih veličin ter parametrov (upornosti, induktivnosti itd.).

Osnovna prednost tega načina je v boljšem izkoristku številskega področja. Po drugi strani pa se bodo digitalni ekvivalenti nazivnih vrednosti posameznih fizikalnih veličin razlikovali.

#### **10.3.5 Realizacija nekaterih splošno uporabnih funkcij**

V tem učbeniku si bomo ogledali realizacijo nekaterih zelo pogostih členov in funkcij na procesnem mikroračunalniku ob uporabi celoštvenske aritmetike (zaporedne številke označujejo bloke na slikah 10. 3 in 10. 4):

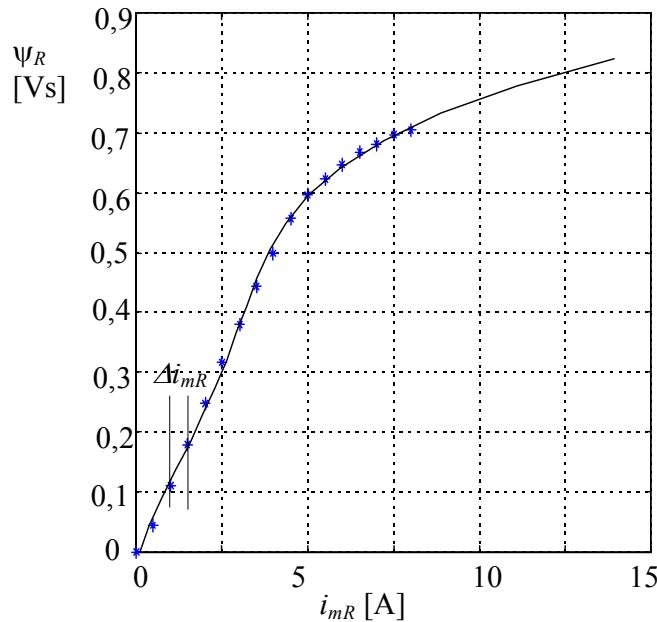
- ① nelinearne funkcije,
- ② trigonometrične funkcije (sinus in kosinus),
- ③ člen prvega reda,
- ④ integralni člen,
- ⑤ diferencialni člen,
- ⑥ PI regulator,
- ⑦ pulzno-širinska modulacija.

##### 10.3.5.1 Primer realizacije nelinearne funkcije

V algoritmu pogosto nastopajo zapletene matematične funkcije, ki jih ne moremo realizirati z majhnim številom aritmetičnih ukazov. Takšen primer, ki sicer ni prikazan na prejšnjih blokovnih shemah, je upoštevanje magnetilne krivulje v algoritmu za regulacijo asinhronskega motorja [2]. V prvem delu krivulje (slika 10. 13), je odvisnost izhodne spremenljivke (fluksa  $\Psi_R$ ) od vhodne veličine (magnetilnega toka  $i_{mR}$ ) skorajda linearna, dokler ne pride do nasičenja.

---

<sup>11</sup> Pomik za 12 bitov v desno in upoštevanje le LSW ali pomik za štiri bite v levo in upoštevanje MSW.



Slika 10. 13: Nelinearna funkcija odvisnosti fluksa od magnetilnega toka

Pri reševanju zgornjega problema lahko načeloma izberemo podobni poti, ki sta opisani v poglavju o realizaciji trigonometričnih funkcij.

#### 10.3.5.1.1 Tabeliranje funkcije

V pomnilniško področje shranimo nekaj normiranih vrednosti izhodne spremenljivke (označene z zvezdico na sliki 10. 13)<sup>12</sup>. Postopek vpisovanja in branja tabeliranih vrednosti je enostaven:

- Z meritvijo smo dobili  $N$  parov točk (koordinat) funkcije  $\Psi_R = \Psi_R(i_{mR})$ , torej bo tabela vsebovala  $N$  podatkov. Korak povečanja vhodne spremenljivke, to je razlika med dvema sosednjima podatkom v tabeli  $\Delta i_{mR}$ , je konstanten. Na  $k$ -ti lokaciji, ki ustreza toku  $i_{mR}(k)$ , se nahaja podatek  $\Psi_R(k)$ , na naslednji ( $k + 1$ ) se nahaja podatek  $\Psi_R(k + 1)$ . Ta ustreza vhodni spremenljivki  $i_{mR}(k + 1)$  ozziroma  $i_{mR}(k) + \Delta i_{mR}$ .
- Pri branju tabeliranih podatkov moramo najprej vhodni podatek  $i_{mR}(k)$  zmnožiti s faktorjem  $N/i_{mR}(N)$ , kjer sta  $N$  število tabeliranih podatkov in  $i_{mR}(N)$  zadnja zajeta vrednost vhodne spremenljivke v tabeli. Rezultat zaokrožimo na celo vrednost. Tako dobljen zmnožek je v bistvu indeks iskanega podatka v tabeli oz. njegov relativni naslov glede na začetek tabele (angl. offset). Za to lahko uporabimo ukaz za prenos z posrednim naslavljanjem s pomočjo naslovnega registra in odmika (glej tudi pogl. 4.3). Tako bo efektivni naslov podatka enak seštevku naslova začetne lokacije tabele (v odmiku: **zactable**) in offset-a (v naslovnu registru **A0**):

```
MOVE (zactable,A0),D0
```

<sup>12</sup> **Pozor!** Vrednosti na sliki ustrezajo realnim vrednostim v koherentnih enotah. Pri manipuliranju s temi vrednostmi v mikroračunalniku jih moramo najprej normirati, kot je to opisano v predhodnih poglavjih.

Vsekakor je zelo malo verjetno, da bodo vhodne vrednosti točno ustrezale tabeliranim abscisnim vrednostim. Zato se je treba odločiti, kateri izmed dveh sosednjih točk je bližja vhodna vrednost in prebrati vsebino njene lokacije. Če menimo, da je napaka prevelika (delitev je pregroba), lahko opravimo linearno interpolacijo (glej naslednje poglavje).

**Primer:**

Najprej bomo normirali vhodne in izhodne vrednosti s slike 10. 13. Predpostavimo, da ustreza magnetilnemu toku 4,8 A in pripadajočemu fluksu 0,584 Vs vrednosti 1000<sub>HEX</sub>, (slika 10. 14). Tabela se začne na naslovu 11f0<sub>HEX</sub>.

| naslov podatka<br>v tabeli (HEX) | vsebina tabele<br>(normirani $\Psi_R$ )<br>v HEX | $i_{mR}$ [A] in pripadajoči<br>fluks [Vs] |
|----------------------------------|--|---|
| 1200                             |  | 4,0 (.....)                               |
| 1202                             | f57  | 4,5 (0,56)                                |
| 1204                             | 1070   | 5,0 (0,60)                                |
| 1206                             | 10fc   | 5,5 (0,62)                                |
| 1208                             | 1188   | 6,0 (0,64)                                |
| 120a                             | 1215   | 6,5 (0,66)                                |
| 120c                             |  | 7,0 (.....)                               |

**Slika 10. 14: Prikaz dela tabele magnetilne krivulje**

#### 10.3.5.1.2 Realizacija nelinearne funkcije s pomočjo polinoma

Nelinearno funkcijo v določenem območju lahko aproksimiramo s polinomom višjega reda. Koeficiente polinoma, ki je najboljši približek krivulje skozi izmerjene točke, je možno zelo enostavno izračunati z enim izmed komercialnih programov (npr. Matlab). Stopnja polinoma je odvisna od oblike aproksimirane funkcije, največja omejitev pri izbiri višje stopnje pa je čas potreben, za izračun polinoma v mikroprocesorju. Polinom je treba za vsako vrednost znova izračunati. Polinom četrtega reda, ki ustreza točkam na nenormirani krivulji s slike 10. 13, kaže naslednja enačba:

$$\begin{aligned}\Psi_R &= a_4 i_{mR}^4 + a_3 i_{mR}^3 + a_2 i_{mR}^2 + a_1 i_{mR}^1 + a_0 = \\ &= 2,18 \cdot 10^{-5} \cdot i_{mR}^4 - 8,24 \cdot 10^{-4} \cdot i_{mR}^3 + 7,71 \cdot 10^{-3} \cdot i_{mR}^2 + 4,28 \cdot 10^{-2} \cdot i_{mR}^1 - 7,04 \cdot 10^{-4}.\end{aligned}$$

Kot vidimo, je za izračun funkcijске vrednosti treba vsakič opraviti veliko operacij seštevanja in množenja (potenciranja in množenja s konstanto), pri čemer sodijo zadnje operacije pri klasičnih mikroprocesorjih med časovno najdaljše. Primernejši od prejšnje oblike je t.i. *Hornerjev zapis*

$$\Psi_R = (((\underbrace{(a_4 i_{mR} + a_3)}_1 \cdot i_{mR} + a_2) \cdot i_{mR} + a_1) \cdot i_{mR} + a_0),$$

1  
 2  
 3  
 4

ki ga enostavno izvedemo na DSP-jih. Ti vsebujejo ukaz za množenje s hkratnim prštevanjem k prejšnjemu rezultatu. Zgornji polinom bi tako lahko izračunali s štirimi ukazi. Nikakor pa ne gre pozabiti, da mora vsak algoritem poleg zgoraj navedenih matematičnih operacij vsebovati še dodatne operacije, ki preprečujejo nezaželena stanja (npr. preverjanja in omejitve zaradi morebitnega presežka v registru itd.). Zato so programerji ta način podajanja funkcije v časovno kritičnih sistemih uporabljali le izjemoma.

#### 10.3.5.2 Primer realizacije trigonometričnih funkcij

Trigonometrične funkcije so zelo pogost element v regulacijskih shemah, ki opisujejo dogajanja v električnih sistemih, kot so npr. omrežno napajana bremena ali regulirani izmenični pogoni. V blokovnih shemah iz našega zogleda jih srečujemo pri transformacijah veličin iz enega pravokotnega koordinatnega sistema v drugi, katerega abscisa je premaknjena za nek kot  $\rho$ . Takšen blok je na sliki 10. 4 označen z ②.

##### 10.3.5.2.1 Uporaba standardnih trigonometričnih funkcij

Pri realizaciji katerekoli funkcije v realnem času v sistemih z zelo majhnimi časovnimi konstantami poskušamo trajanje algoritma zmanjšati na najkrajši možni čas, zato uporaba trigonometričnih funkcij v višjih jezikih (npr. C, Pascal, Basic...) v našem primeru ne pride v poštev. Operacije so del že obstoječih knjižnic programa (npr. `sin(a)` v C jeziku). Čeprav je uporaba teh funkcij elegantna, pa časovno ni optimalna, kar je splošno znan problem višjih jezikov. Le v primeru, ko imamo opravka z izredno hitrimi procesorji (npr. DSP-ji) ali počasnimi procesi, si lahko dovolimo njihovo uporabo. Največkrat mora programer funkcijo, ki ustreza tipičnim zahtevam njegovega programa, zelo skrbno programirati v zbirnem jeziku. Tudi najsodobnejši prevajalniki dosežejo le (ali že) 80 % časovnega optimiranja, ki ga doseže izkušeni programer.

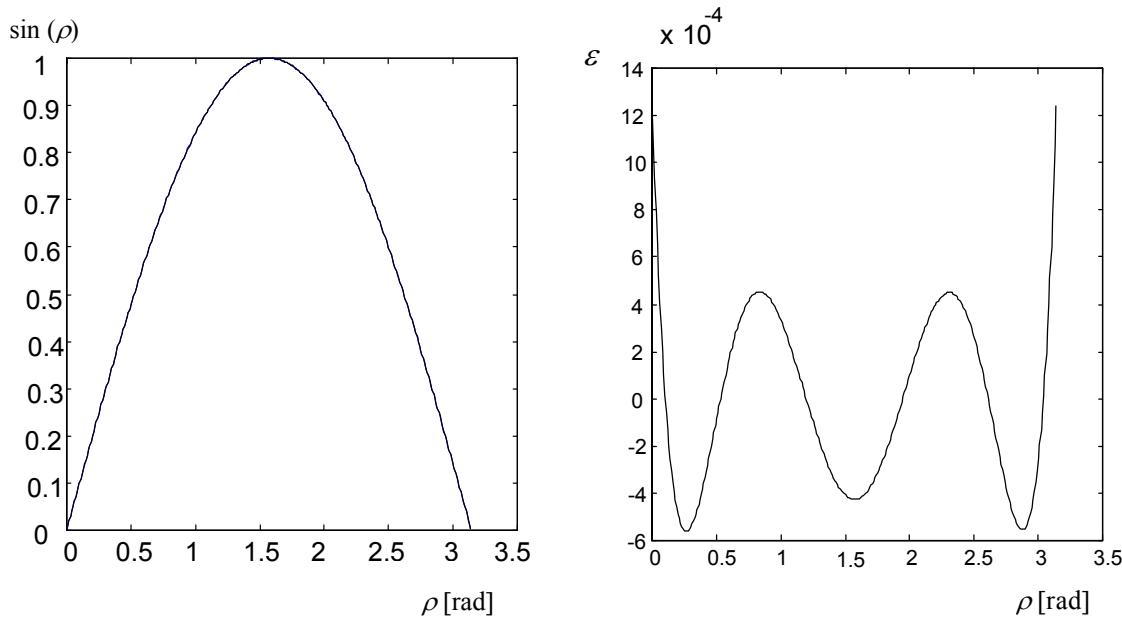
##### 10.3.5.2.2 Zapis sinusne funkcije v obliki polinoma

Vsako trigonometrično funkcijo lahko razvijemo v neskončno vrsto, večinoma pa zadostuje že nekaj prvih členov. Ta način zpisa ni optimalen na DSP-ju, ker vsebuje le sode ali lihe potence. Zato DSP ukaza za množenje in hkratno seštevanje iz poglavja 10.3.5.1.2 ne moremo izkoristiti v največji možni meri.

Če se že odločimo za vsakokratni izračun trigonometrične funkcije, raje uporabimo zapis v obliki polinoma. Npr. vsako od polperiod sinusne funkcije izračunamo s posebnim polinomom:

$$\sin \rho = \begin{cases} (((((0,0372 \cdot \rho) - 0,2338) \cdot \rho + 0,0544) \cdot \rho + 0,9826) \cdot \rho + 0,0013, & \text{če } \rho \geq 0 \\ (((((0,0372 \cdot \rho) + 0,2338) \cdot \rho + 0,0544) \cdot \rho - 0,9826) \cdot \rho + 0,0013, & \text{če } \rho < 0 \end{cases}$$

Slika 10. 15 kaže sinusno funkcijo, ki smo jo izračunali s pomočjo zgornje vrste ter razliko (pogrešek) med tako dobljenimi vrednostmi funkcije in pravo vrednostjo sinusa v odvisnosti od kota.



**Slika 10. 15: Potek sinusne funkcije, ki je bila izračunana s pomočjo končne vrste in odstopanje od tiste neskončne vrste**

#### 10.3.5.2.3 Tabelarično podajanje trigonometričnih funkcij

Najpogosteje uporabljan način izračuna trigonometričnih funkcij v časovno zahtevnih sistemih je metoda tabelaričnega podajanja vrednosti, podobno kot pri nelinearnih funkcijah iz pogl. 10.3.5.1.1. Velja omeniti nekaj koristnih napotkov:

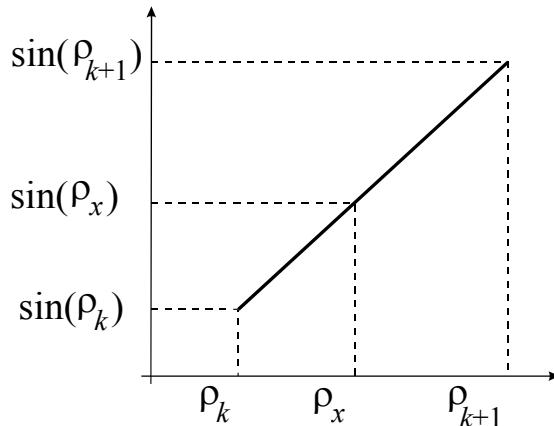
##### Število tabeliranih vrednosti:

Pričakovati je, da je natančnost podajanja funkcije s pomočjo tabele odvisna od števila elementov tabele (ločljivost). Po drugi strani bo preveliko število elementov zasedlo preveč prostora v pomnilniku. Kompromis lahko dosežemo tako, da tabeliramo le vrednosti sinusa, ki ustrezajo četrtini periode, saj so vrednosti v ostalih treh kvadrantih zrcalne podobe prvega kvadranta s pozitivnim ali z negativnim predznakom (drugi in četrji kvadrant) oziroma kar negativne vrednosti funkcije za prvi kvadrant (tretji kvadrant). Seveda bo celoten algoritem, ki ugotavlja kvadrant podanega kota in relacijo s tabeliranim kotom v prvem kvadrantu, nekoliko daljši.

Število podatkov nadalje zmanjšamo (večji korak med dvema vrednostima) tako, da kote, ki niso zajeti v tabeli, izračunamo z linearno interpolacijo (10. 3)

$$\sin(\rho_x) = \sin(\rho_k) + \frac{\rho_x - \rho_k}{\rho_{k+1} - \rho_k} \cdot (\sin(\rho_{k+1}) - \sin(\rho_k)), \quad (10.3)$$

kjer sta s  $k$  in  $k+1$  označeni dve zaporedni tabelirani vrednosti,  $\rho_x$  pa je kot, katerega sinus želimo izračunati (slika 10. 16).



**Slika 10. 16: Linearna interpolacija za vrednosti med dvema podatkoma v tabeli**

Število podatkov v tabeli (ločljivost) je med ostalim odvisno od načina normiranja, navadno pa je enako  $n$ -ti potenci števila 2, npr.  $2^8 = 256$ ,  $2^9 = 512$ ,  $2^{10} = 1024$ .

#### Normiranje kota in vrednosti sinusa:

*Normiranje kota:*

Vrednosti kota in pripadajočih sinusov so normirane (glej podpoglavlja o zajemanju kota rotorja 10.3.4.4.2 in integratorju 10.3.5.4). Koti imajo le pozitivne vrednosti ( $0 - 2\pi$ ), zato logike dvojiškega komplementa pri normiranju zaradi boljše ločljivosti načeloma ne upoštevamo. Tako bodo normirane vrednosti **nepredznačene**, kar pri 16-bitnem normiranju pomeni vrednosti od  $0000_{HEX}$  do  $ffff_{HEX}$  (format f0.16). Ena od prednosti takega formatiranja se kaže v samodejnem izvajanju operacije "modul  $2\pi$ ": ob povečanju zadnje nepredznačene vrednosti  $ffff_{HEX}$  (kot  $2\pi/65535$ ) bo naslednja vrednost  $0_{HEX}$  (kot 0).

Zaradi poenostavitev bomo v nadaljevanju predpostavili, da tabeliramo sinusne vrednosti za celotno periodo ( $0 - 2\pi$ ) in ne le četrtino. V tabelo seveda ne bomo vpisali sinusnih vrednosti za vseh 65536 vrednosti kota, ampak izberemo bistveno manjše število, npr. eno izmed predlaganih v prejšnji točki, ki je potenza števila 2. Prednost tega pristopa je v enostavni prilagoditvi kota dolžini tabele s pomočjo operacije pomika ("shift") v desno (deljenje z  $2^n$ ). Pri izbiri  $2^8 = 256$  podatkov (od 0 do  $2\pi$ ) moramo pred branjem tabele normirano vrednost kota premakniti za osem bitov v desno ( $2^{16}/2^8 = 2^8$ ), pri  $2^9 = 512$  podatkih za sedem bitov ( $2^{16}/2^7 = 2^9$ ) itd.

V našem zgledu pa bomo zaradi lažje obravnave in kompatibilnosti s formati ostalih veličin kote normirali v formatu f4.12, pri čemer bo kotu  $2\pi$  ustrezala vrednost "1" ali  $1000_{HEX}$ . Štirje

zgornji biti bodo vedno nič, kar dosežemo z maskiranjem v modul  $2\pi$  (glej pogl. 10.3.4.4.2)<sup>13</sup>. Tej operaciji sledi še prilagoditev območja kota dolžini tabele, kar dosežemo na enak način, kot smo to naredili prej: s premikom vrednosti kota za določeno število bitov v desno, odvisno od dolžine tabele.

*Normiranje vrednosti sinusa:*

Zaloga vrednosti sinusa ali kosinusa je med -1 in +1 (normirano “-1” do “+1”), zato načeloma tabelirane vrednosti določimo z upoštevanjem dvojiškega komplementa ( $8000_{HEX} = -32768_{DEC}$  do  $7FFF_{HEX} = 32767_{DEC}$  pri 16-bitnem normiraju). Tako v celoti izkoristimo 16-bitni obseg.

Če pa želimo obdržati format f4.12, bo normirani vrednosti “-1” ustrezal podatek  $f000_{HEX}$ , “+1” pa  $1000_{HEX}$ .

**Primer:**

V tabelo smo shranili 512 ( $2^9$ ) vrednosti celotne periode sinusne funkcije. Na začetni, “ničti” lokaciji, ki ustreza kotu  $0 \cdot 2\pi / 512$  se nahaja vrednost sinusne funkcije za ta kot (0), na naslednji lokaciji (kot  $1 \cdot 2\pi / 512$ ) se nahaja vrednost  $\sin(0,01227) = 0,0123 \Leftrightarrow “0,0123” \cdot 4096_{HEX} = 32_{DEC} = 32_{HEX}$ . Tretja lokacija (kot  $2 \cdot 2\pi / 512$ ) vsebuje vrednost  $65_{HEX} (“0,0245”) \cdot 4096_{DEC} = 101_{DEC}$ , zaradi  $\sin(0,02456) = 0,02455$ . Maksimalna pozitivna vrednost na 127. lokaciji (kot  $127 \cdot 2\pi / 512 \Leftrightarrow \pi/2$ ) pa je enaka  $1000_{HEX}$ <sup>14</sup>. Maksimalna negativna vrednost sinusne funkcije se nahaja na 384. lokaciji in je enaka  $f000_{HEX} \Leftrightarrow “-1”$ . Nekatere vrednosti kaže tabela 10.4. Zaradi omejene ločljivosti so vrednosti v bližini amplitude, kjer je spremembra najmanjša, enake.

| Naslov podatka<br><b>(Začetni naslov tabele +)</b> | Kot<br>(rad) | Vsebina lokacije<br>(HEX) | Vsebina lokacije<br>(DEC) | Vrednost<br>sinusa |
|--|--------------|---------------------------|---------------------------|--------------------|
| 0  | 0            | 0                         | 0                         | 0                  |
| 1  | 0,01223      | 32                        | 50                        | 0,0122             |
| 2  | 0,0245       | 65                        | 101                       | 0,0245             |
| ...  |              |                           |                           |                    |
| $127_{DEC} = 7f_{HEX}$                             | 1,559        | 1000                      | 4096                      | 0,9999             |
| ...  |              |                           |                           |                    |
| $383_{DEC} = 17f_{HEX}$                            | 4,7          | f000                      | -4096                     | -0,9999            |
| $384_{DEC} = 180_{HEX}$                            | 4,7123       | f000                      | -4096                     | -1                 |
| $385_{DEC} = 181_{HEX}$                            | 4,7247       | f000                      | -4096                     | -0,9999            |
| ...  |              |                           |                           |                    |
| $511_{DEC} = 1ff_{HEX}$                            | 6,27         | ffce                      | -50                       | -0,0123            |

**Tabela 10.4: Nekatere vrednosti tabelirane funkcije sinusa v formatu f4.12**

<sup>13</sup>Če imamo opravka z inkrementalnim dajalnikom z  $8192_{DEC} = 2000_{HEX}$  pulzi (4·2048; pogl. 10.3.4.4.2), dosežemo pretvorbo v format f4.12 (po maskirjanju v modul  $2\pi$ ) s premikom za en bit v desno.

<sup>14</sup>Ponovno se moramo spomniti, da je pri dvojiškem komplementu negativni obseg za eno vrednost večji od pozitivnega.

#### 10.3.5.2.4 Kratka primerjava opisanih načinov izračuna trigonometrijskih funkcij

Pri klasičnih mikroprocesorjih ima tabelarično podajanje funkcij prednost pred razvojem v vrsto zaradi manjšega števila množenj, ki zahtevajo bistveno več procesorskega časa. Pri DSP-jih pa je stanje popolnoma drugačno: operacije množenja zahtevajo približno enako število ukaznih ciklov kot ostale operacije. Zato je pri njih čas izvajanja algoritma za izračun tabelarično podane trigonometrične funkcije daljši od časa, ki je potreben za razvoj v vrsto na DSP TMS320F240 (tabela 10.5) [28].

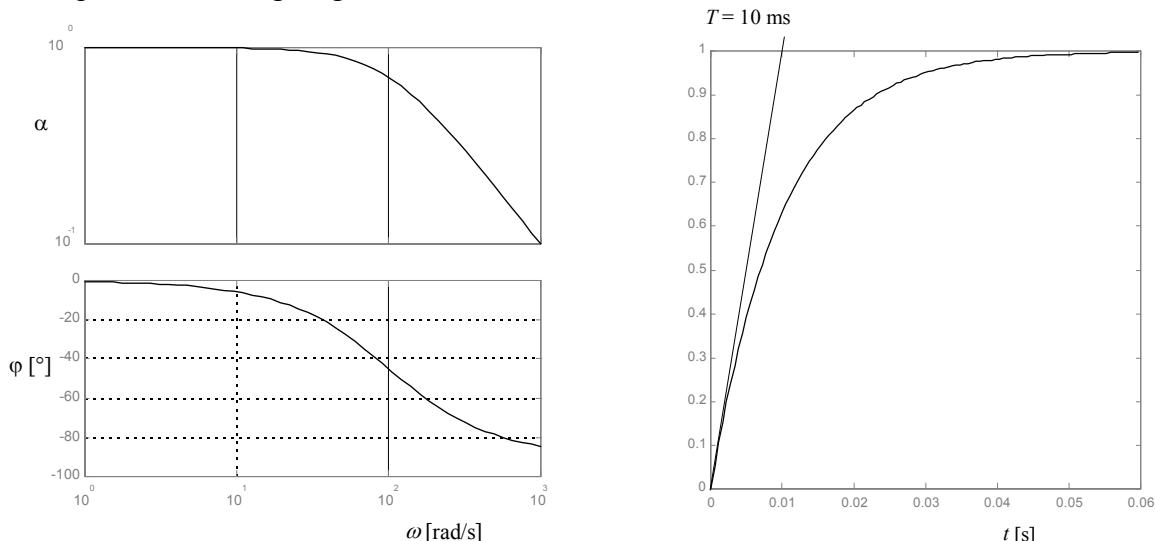
| Čas izvajanja funkcije (v $\mu$ s) |  |                |
|------------------------------------|--|----------------|
| Funkcija                           | Tabelarično podajanje in interpolacija | Razvoj v vrsto |
| <b>sin</b>                         | 1,75                                   | 1,35           |
| <b>cos</b>                         | 1,90                                   | 1,85           |

**Tabela 10.5:** Primerjava časa izvajanja trigonometričnih funkcij v zbirnem jeziku s pomočjo razvoja v vrsto in tabele (DSP)

#### 10.3.5.3 Primer realizacije člena prvega reda

##### 10.3.5.3.1 Opis lastnosti člena prvega reda

Člen prvega reda (angl. first order lag) pogosto srečamo v regulacijskem programu, ki ga je treba izvajati v realnem času (blok ③ na sliki 10. 4) [2,4]. Iz njegove frekvenčne karakteristike (slika 10. 17) sta razvidna amplitudni in fazni potek v odvisnosti od frekvence vhodnega signala. Drugi del slike kaže časovni odziv člena prvega reda na enotino funkcijo (t.i. prehodna funkcija). Ob faktorju jačanja  $K = 1$  (glej tudi (10. 4)) se člen lahko uporablja kot t.i. nizkopasovni filter, saj v področju višjih frekvenc (nad frekvenco  $\omega_0 = T^{-1}$ ), občutno duši amplitudo vhodnega signala.



**Slika 10. 17:** Frekvenčna karakteristika in prehodna funkcija člena prvega reda ( $K = 1$ ,  $T = 10 \text{ ms}$  oz.  $\omega_0 = 100 \text{ rad/s}$ )

Člen prvega reda opisuje diferencialna enačba prvega reda

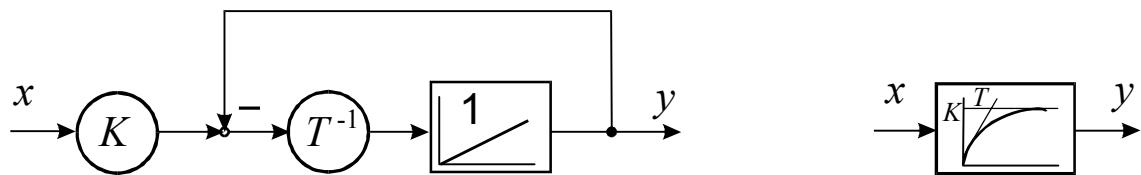
$$T \frac{dy(t)}{dt} + y(t) = K x(t), \quad (10.4)$$

kjer je  $x$  vhodna,  $y$  pa izhodna veličina ter  $K$  ojačenje in  $T$  časovna konstanta sistema.

Prenosna funkcija člena prvega reda (razmerje med izhodnim in vhodnim signalom v frekvenčnem prostoru) je določena z izrazom

$$F(p) = \frac{y(p)}{x(p)} = \frac{K}{1 + pT} = \alpha e^{-j\varphi}. \quad (10.5)$$

Na sliki 10. 18 sta možna načina grafičnega prikaza bloka člena prvega reda.



**Slika 10. 18: Grafična prikaz člena prvega reda**

#### 10.3.5.3.2 Diferenčna enačba člena prvega reda

Člen prvega reda v mikroračunalniku realiziramo z numeričnim reševanjem diferencialne enačbe (10. 4). Glede na diskretno naravo delovanja procesnega mikroračunalnika, ki ciklično, s periodo vzorčenja  $T_V$ , izvaja regulacijski algoritem, moramo diferencialno enačbo najprej diskretizirati. Na ta način dobimo t.i. *diferenčno enačbo*. Obstaja več različnih pristopov k reševanju differenčnih enačb. V časovno kritičnih regulacijskih sistemih se verjetno najbolj splača najhitrejša metoda, ki ni nujno tudi najbolj natančna.

Diferenčno inačico enačbe (10. 4) dobimo tako, da izračunamo odvod v končno kratkem intervalu, to je času vzorčenja  $T_V$ . Velja:

$$\lim_{\Delta t \rightarrow 0} \frac{\Delta y}{\Delta t} = \lim_{T_V \rightarrow 0} \frac{\Delta y}{T_V} = \lim_{T_V \rightarrow 0} \frac{y(k) - y(k-1)}{T_V} = \frac{dy}{dt}, \quad (10.6)$$

kjer je  $\Delta y$  razlika (*diferenca*) vrednosti izhodne spremenljivke v trenutnem ( $k$ ) in prejšnjem vzorcu ( $k-1$ ), ob zelo kratkem intervalu vzorčenja  $T_V$ . Diferenčna enačba člena prvega reda je potem takem:

$$\frac{T}{T_V} (y(k) - y(k-1)) + y(k) = K x(k). \quad (10. 7)$$

Že iz (10. 6) je razvidna zahteva po čim krajšem času vzorčenja. Ta pa je po drugi strani omejen s trajanjem obdelave celotnega regulacijskega programa, ki ga je treba izvršiti v tem intervalu, ter hitrostjo procesorja. V praksi je velikost  $T_V$  odvisen tudi od časovnih konstant sistema (v našem primeru  $T$ ), saj je pri zelo velikih časovnih konstantah (npr. nekaj minut) nesmiselno vztrajati pri zelo majhnih intervalih vzorčenja (npr. nekaj ms). Običajno je čas vzorčenja vsaj desetkrat (po možnosti tudi nekaj desetkrat) krajiši od časovne konstante člena, npr.  $T_V = 1$  ms pri  $T = 40$  ms.

Rešitev diferenčne enačbe (10. 7) lahko zapišemo v obliki

$$y(k) = \frac{T_V K}{T + T_V} x(k) + \frac{T}{T + T_V} y(k-1) \quad (10. 8)$$

To je *rekurzivna* enačba, v kateri je vrednost izhoda v tekočem vzorcu  $y(k)$  odvisna od tekoče vrednosti vhoda  $x(k)$  ter vrednosti izhoda  $y(k-1)$ , ki smo jo že izračunali v prejšnjem vzorcu. Začetni pogoj za  $y(0)$  je običajno 0. Diferenčno enačbo lahko zapišemo tudi v diskretnem  $z$  prostoru:

$$F(z) = \frac{bz}{z-a}, \quad (10. 9)$$

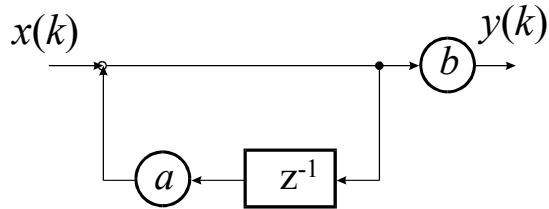
kjer sta v našem primeru

$$a = \frac{T}{T + T_V} \quad \text{in} \quad b = \frac{T_V K}{T + T_V}. \quad (10. 10)$$

Pri tem je  $z^{-1}$  je *operator zakasnitve* za eno periodo vzorčenja; glej tudi [2, 13]:

$$y(k-1) = z^{-1} y(k).$$

Iz tega sledi tudi blokovna shema s slike 10. 19.



Slika 10. 19: Blokovna shema diskretnega člena prvega reda

### 10.3.5.3.3 Praktična realizacija člena prvega reda v procesnem mikroračunalniku

Kot primer vzemimo realizacijo člena prvega reda s parametrom  $K = 1$ ,  $T = 50$  ms, ki ga bomo realizirali znotraj intervala vzorčenja  $T_V = 1$  ms. Enačba 10. 8 se sedaj glasi:

$$y(k) = "0,0196" \cdot x(k) + "0,9804" \cdot y(k-1).$$

Konstante bomo zaradi združljivosti z ostalimi veličinami in enostavnosti zapisali v formatu f4.12, čeprav bi bil v tem primeru primernejši format f1.15, s katerim bi polno izkoristili ločljivost mikroprocesorja. Tako je prva konstanta 50<sub>HEX</sub>, druga pa fb0<sub>HEX</sub>.

Prekinitven podprogram za realizacijo člena prvega reda v MC68332 je zelo kratek:

```
*****
***          CLEN PRVEGA REDA
***      vhod na naslovu (x), izhod na naslovu (y)
***      konstante k1=T/(T+Tv), k2=K*Tv/(T+Tv),
***          ****
*****          ****
clr.l      d3          *** brisi registra
clr.l      d1          *** za vsak primer

move.w    (x),d3        *** vhod v ptl x(k) ***
move.w    (y),d1        *** stara vrednost izhoda y(k)

mul.s.w   #k1,d1        *** mnozenje s k1=T/(T+Tv)
mul.s.w   #k2,d3        *** mnozenje s k2=K*Tv/(T+Tv)

add.l     d1,d3        *** y(k)=k2*x(k)+k1*y(k-1)

asl.l     #4,d3        *** pomik za 4 bite v levo
                      *** (rezultat v zgornji besedi)
swap.w    d3            *** postavi ga v spodnjo besedo
move.w    d3,(y)         *** shrani y(k)
```

Pred tem smo določili konstante ter rezervirali pomnilniški prostor za spremenljivke (pogl. 9.1.2):

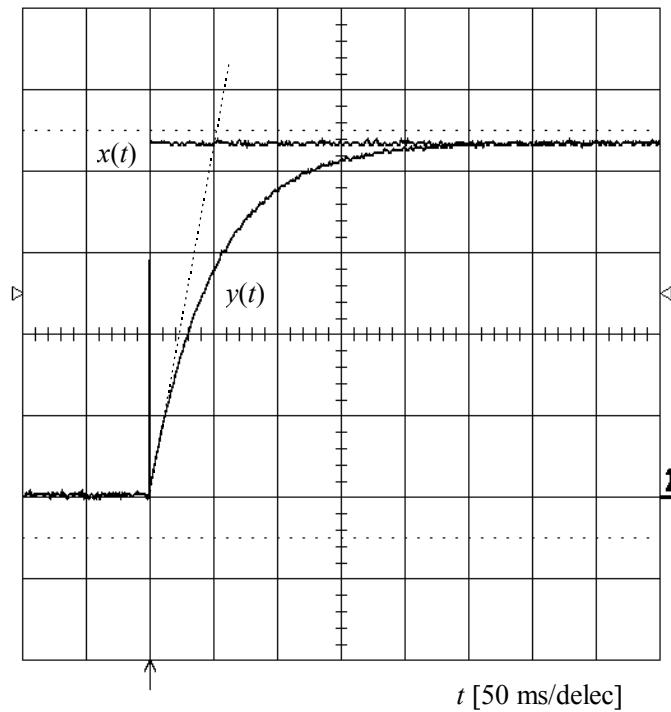
```
k1    equ  $fb0          *** definicija konstante k1
k2    equ  $50            *** definicija konstante k2

*** rezervacija pomnilniškega prostora

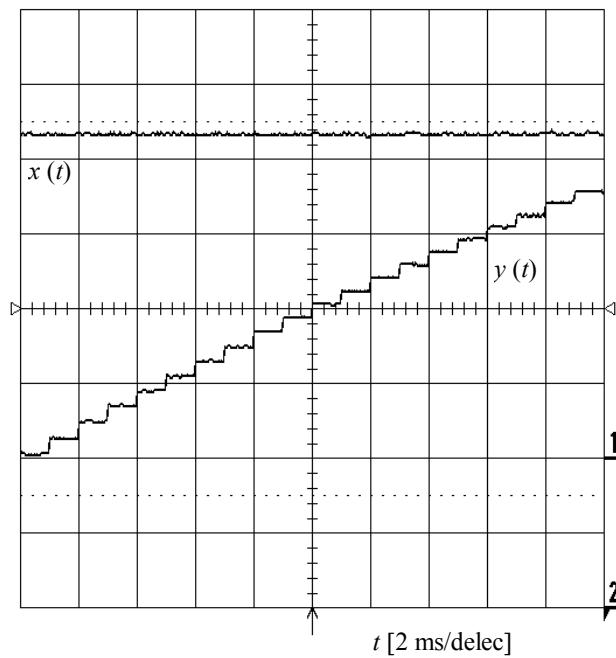
x    ds.w 1              *** rezerviraj prostor v dolzini
y    ds.w 1              *** ene besede za x in y
```

Zgornje spremenljivke moramo v fazi inicializacije postaviti v stanje 0.

Slika 10. 20 kaže oscilogram prehodne funkcije člena prvega reda. Na sliki 10. 21 je prikazan isti odziv ob zmanjšani časovni bazi, kjer se razločno vidi amplitudna in časovna diskretizacija.



**Slika 10. 20:** Oscilogram prehodne funkcije člena prvega reda



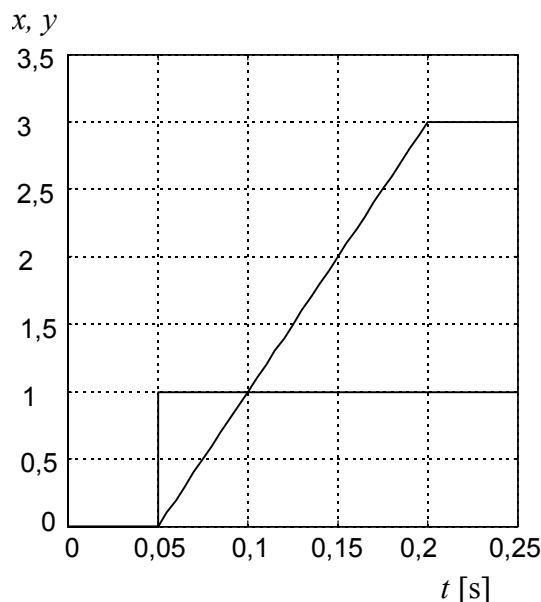
**Slika 10. 21:** Povečani oscilogram s prejšnje slike

### 10.3.5.4 Primer realizacije integralnega člena

Integrator je tudi pogost element v regulacijski shemi (npr. blok ⑤ na sliki 10. 4). Pri razvoju ustreznega podprograma izhajamo iz enačbe v časovnem prostoru

$$y(t) = \frac{1}{T_i} \int_0^t x(\tau) d\tau = K_i \int_0^t x(\tau) d\tau, \quad (10.11)$$

kjer je  $T_i$  časovna konstanta integralnega člena oz.  $K_i$  njena obratna vrednost. Izvajanje integratorskega podprograma mikroprocesorja se začne v trenutku  $t = 0$  s. Časovna konstanta je čas, v katerem doseže izhodna veličina spremembo v velikosti svoje osnovne vrednosti, če deluje na vhodu veličina s konstantno osnovno vrednostjo (slika 10. 22) [4].



Slika 10. 22: Prehodna funkcija integratorja s  $T_i = 50$  ms in vhodna enotina stopnica

Čeprav obstaja več načinov pretvorbe zgornje enačbe v diferenčno obliko, bomo tukaj uporabili najenostavnnejšo (Eulerjevo ali pravokotno metodo). Pri končno majhnem času  $T_V$  se enačba (10. 11) glasi:

$$y(k) = K_i \sum_{i=0}^k x(i) T_V = K \sum_{i=0}^k x(i), \quad (10.12)$$

kjer je  $K = K_i \cdot T_V$ .

Enačbo lahko zapišemo tudi v rekurzivni obliki, saj velja

$$y(k) = Kx(k) + K \sum_{i=0}^{k-1} x(i) = Kx(k) + y(k-1).$$

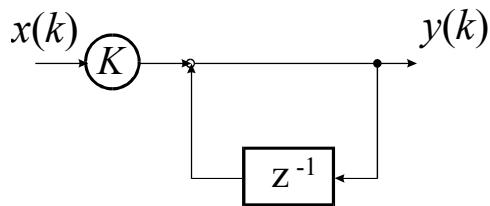
Ob uporabi operatorja zakasnitve, lahko zapišemo

$$y(k) = Kx(k) + z^{-1}y(k),$$

iz tega sledita prenosna funkcija

$$F(z) = \frac{K \cdot z}{z - 1}$$

ter blokovna shema na sliki 10. 23.



**Slika 10. 23: Blokovna shema diskretnega integralnega člena**

V mikroprocesorju integralni člen realiziramo zelo enostavno. Pri tem je najpomembnejša konstanta  $K$ , ki je odvisna od integracijske časovne konstante in časa vzorčenja.

**Primer:**

V mikroprocesorju želimo realizirati integrator s časovno konstanto 50 ms pri  $T_V = 1$  ms. Iz prej povedanega sledi, da mora pri vzbujanju z enotino stopnico (1000<sub>HEX</sub> pri f4.12) izhod doseči vhodno vrednost v času  $t = 50\text{ms}$ , to je po petdesetih vzorcih. Konstanta  $K$  iz (10. 12) je torej  $4096/50 \approx 82_{\text{DEC}} = 52_{\text{HEX}}$ .

Pri realizaciji integratorja moramo biti še posebej pozorni na omejevanje izhodne veličine (slika 10. 22). Izhodna vrednost integratorja namreč vseskozi narašča (ali upada), dokler ne pride do spremembe predznaka vhoda (izhodna vrednost začne upadati) ali dokler le-ta ne pade na vrednost nič (izhodna vrednost se ustali). Pri relativno kratkih časovnih konstantah integratorja obstaja realna nevarnost, da integrator doseže skrajno mejo možnega zapisa v registru.

Sledi program za realizacijo integralnega člena iz zgornjega primera v MC68332 in oscilogram odziva (brez inicializacijskega dela):

```
*****
***          INTEGRALNI CLEN
***
***      vhod na naslovu (x), izhod na naslovu (y)
***      vmesni integral (yvmes) v formatu f8.24
***          konstanta ojacenja: k
***

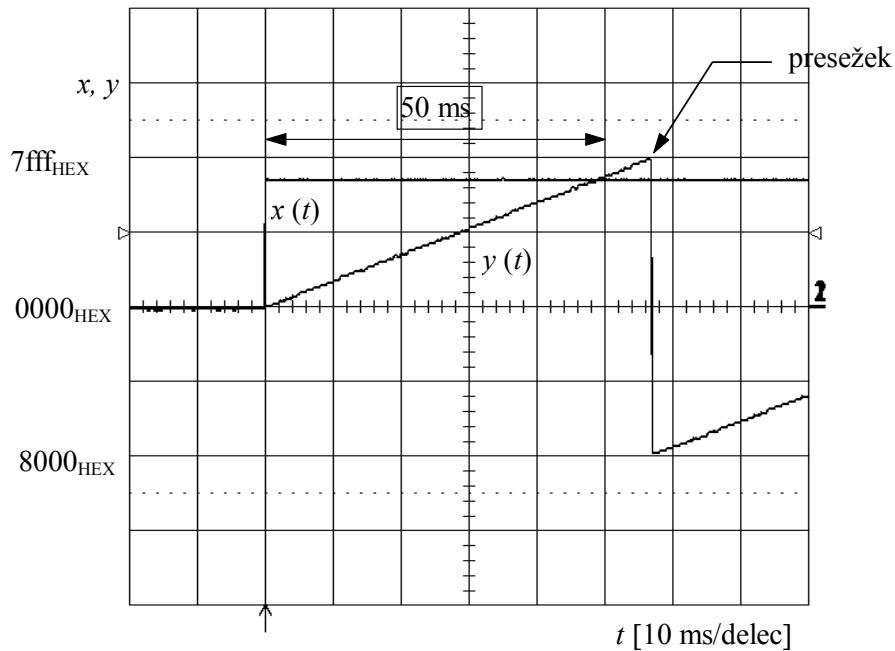
*****
```

|         |            |                                 |
|---------|------------|---------------------------------|
| clr.l   | d1         | *** brisi registra              |
| clr.l   | d0         | *** za vsak primer              |
| move.w  | (x),d1     |                                 |
| muls.w  | #k,d1      | *** mnozi s konstanto k = 82    |
| move.l  | (yvmes),d0 |                                 |
| add.l   | d1,d0      | *** yvmes = yvmes + x*k         |
| move.l  | d0,(yvmes) | *** PAZI! Vmesna vrednost       |
| (yvmes) |            | *** se vedno ni premaknjena za  |
| l2      |            | *** bitov v desno oz. 4 v levo! |
| asl.l   | #4,d0      | *** sele sedaj poravnava v      |
| f4.l2.  |            | *** Rezultat v zgornji besedi!  |
| swap.w  | d0         | *** postavi ga v spodnjo besedo |
| move.w  | d0,(y)     | *** Dejanski izhod v f4.l2      |

V programu razlikujemo dve vrednosti integrala. Prva (**yvmes**) je seštevek produktov s konstanto v 32-bitnem formatu f8.24 ( $f4.12 \cdot f4.12 = f8.24$ ). Na ta način zmanjšamo pogrešek pri kontinuiranem seštevanju, saj računamo vseskozi z 32-bitno ločljivostjo. Šele pred prenosom na lokacijo dejanske izhodne vrednosti (**y**) opravimo poravnavo za 4 bite v levo ter upoštevamo besedo z najmanjšo težo. Zaradi tega moramo za spremenljivko **yvmes** v RAM-u rezervirati prostor 4 byte:

**yvmes**      **ds.l**      **1**

V zgledu smo namenoma spustili del za omejevanje integratorja zaradi ilustracije možnih negativnih posledic. Slika 10.24 kaže rezultat algoritma.



**Slika 10. 24: Oscilogram prehodne funkcije integratorja v realnem času**

Izhodna veličina doseže vrednost vhodne veličine (tukaj  $7000_{\text{HEX}}$ ) po času  $T_V = 50 \text{ ms}$  ter nadaljuje z naraščanjem, ker je vhodna veličina še vedno pozitivna. Ko doseže maksimalno pozitivno vrednost 16-bitnega registra ( $7fff_{\text{HEX}}$ ), pride do presežka, zato izhodna vrednost integratorja nadaljuje svoj potek z negativno vrednostjo ( $8000_{\text{HEX}}$ ). To ne ustreza realnemu integratorju in pripelje do nepredvidljivega obnašanja celotne regulacijske proge!

#### 10.3.5.5 Primer realizacije diferencialnega člena

Diferencialni (D) člen (npr. ⑥ na sliki 10. 3) deluje kot ojačevalnik visokofrekvenčnih motenj, zato je eden najmanj zaželenih elementov vsakega regulacijskega sistema. To še posebej velja za diskrete sisteme [13]. V primerih, ko je diferencialni člen neizogiben (npr. za določanje kotne hitrosti rotorja iz merjenega kota  $\omega = d\theta/dt$ ), mu navadno dodamo še nizkopasovni filter. Moramo se zavedati, da s tem namenoma poslabšamo dinamično karakteristiko diferencialnega člena, ki je ojačevalnik spremembe vhodne veličine, torej zaznava tendenco poteka. Zato je izbira časovne konstante filtra (konstanti  $a$  in  $b$  v (10. 9) oz. (10. 10)) kompromis med motilno frekvenco in pričakovano dinamiko sistema.

Realizacija enostavnega diferencialnega člena sledi iz (10. 6):

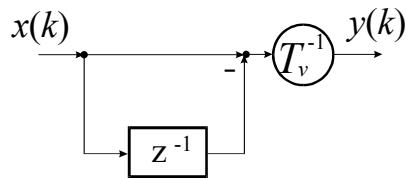
$$\frac{dy}{dt} \approx \frac{y(k) - y(k-1)}{T_V}$$

Prenosno funkcijo dobimo z uvedbo operatorja zakasnitve:

$$\frac{y(z) - z^{-1}y(z)}{T_V} = x(z) \Rightarrow$$

$$F(z) = \frac{y(z)}{x(z)} = \frac{z-1}{T_V z} \quad (10.13)$$

Blokovna shema je prikazana na sliki 10. 25.

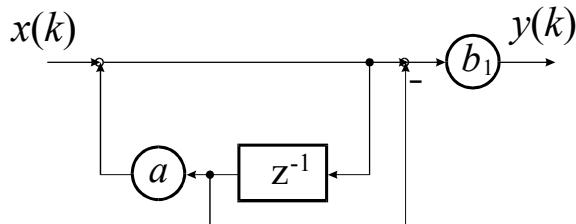


*Slika 10. 25: Blokovna shema diskretnega diferencialnega člena*

Povedali smo že, da pri praktični realizaciji D členu običajno dodamo še nizkopasovni filter. Iz (10. 9) in (10. 13) sledita prenosna funkcija tega člena in blokovna shema.

$$F(z) = \frac{bz}{z-a} \cdot \frac{z-1}{T_V z} = \frac{b}{T_V} \frac{z-1}{z-a} = b_1 \frac{z-1}{z-a},$$

kjer je  $b_1 = b/T_V = K/(T+T_V)$  (glej tudi(10. 10)).



*Slika 10. 26: Prenosna funkcija filtriranega diferencialnega člena*

**Primer:**

Opisali bomo enostavno realizacijo odvajanja kota, ki smo ga zajeli z inkrementalnim dajalnikom (⑥ na sliki 10. 3 in pogl. 10.3.4.4.2) in hkratno prilagajanje normiranim vrednostim. Nizkopasovni filter v tem prikazu ni zajet.

Polni vrtljaj rotorja ustreza  $8192_{DEC}$  ( $2000_{HEX}$ ) pulzom dajalnika. Čas vzorčenja hitrosti (superponirana regulacijska zanka) je daljši od osnovnega intervala vzorčenja (npr.  $100 \mu s$ ), saj so tudi mehanske konstante bistveno večje od električnih. V našem primeru bomo upoštevali  $T_V = 3$  ms. Nazivna hitrost  $148,7 \text{ rad}^{-1}$  (ali  $1420 \text{ min}^{-1}$ ; tabela 10.2) je normirana na nazivno osnovno frekvenco statorskoga navitja (50 Hz ali  $314,159 \text{ rad/s}$ ), ki ji ustreza digitalna vrednost  $1000_{HEX}$  v formatu f4.12. Torej ustreza nazivni hitrosti vrednost "0,473" ali  $1939_{DEC} = 793_{HEX}$ . V eni sekundi se pri nazivni hitrosti rotor zavrti  $1420/60 = 23,667$ -krat, v enem intervalu vzorčenja kota pa naredi  $23,667 \cdot 0,003 = 0,071$  vrtljaljev. To ustreza  $0,071 \cdot 8192 = 582_{DEC} = 246_{HEX}$  pulzom inkrementalnega dajalnika ( $\Delta y$  iz diferenčne enačbe

(10. 6)). Torej bo pri normirani nazivni hitrosti ( $1939_{DEC} = 793_{HEX}$ ) razlika preštetih pulzov inkrementalnega dajalnika med dvema vzorčenjima enaka  $582_{DEC}$ . Iz tega sledi, da bomo za meritev poljubne hitrosti pomnožili število preštetih pulzov s faktorjem  $1939_{DEC}/582_{DEC} = "3,3316"$ , kar v formatu f4.12 pomeni množenje s številom  $13646_{DEC} = 354e_{HEX}$ !

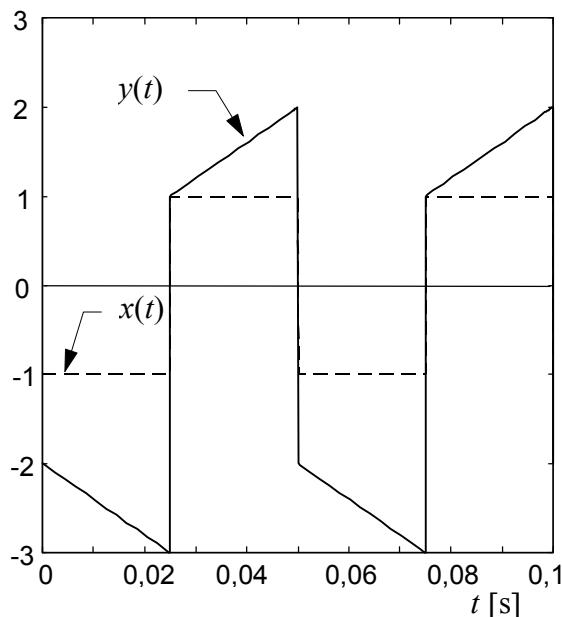
#### 10.3.5.6 Primer realizacije PI člena

PI (7) je tip regulatorja, ki ga najpogosteje srečujemo v zahtevnejših elektromehanskih sistemih. Gre v bistvu za hkratno delovanje (seštevek) integralnega in proporcionalnega člena. S tem dosežemo dobro dinamiko in odpravimo statični pogrešek. Prenosna funkcija PI regulatorja v Laplaceovem prostoru je [4]:

$$F(p) = K_p + \frac{1}{T_i p} = K_p \frac{T_i p + 1}{T_i p} = K_p \frac{T_{ip} p + 1}{T_{ip} p}, \quad (10. 14)$$

kjer je  $T_{ip} = K_p T_i$ .

Slika 10. 27 kaže odziv PI člena na simetričen vlak pulzov. Ojačenje proporcionalnega člena je 2, časovna konstanta integratorja  $T_i$  pa 25 ms, začetna vrednost vhodnega signala je enaka nič.

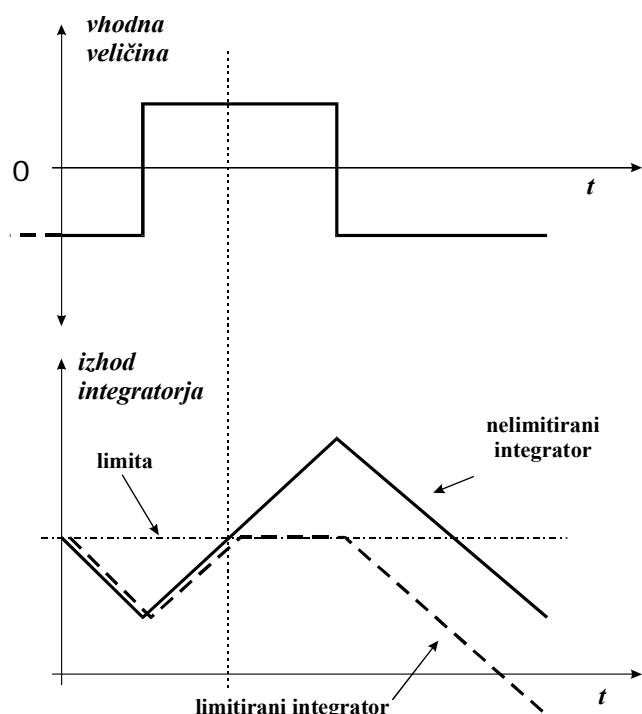


*Slika 10. 27: Odziv PI člena na simetričen vlak pulzov (normirano)*

Glavni problem pri programiranju PI člena je usklajevanje limit. Do presežka področja registra pride bodisi v posameznih vejah PI člena (v proporcionalnem delu pri velikem ojačenju ali veliki vrednosti vhoda; v integralnem delu pri majhnih časovnih konstantah,

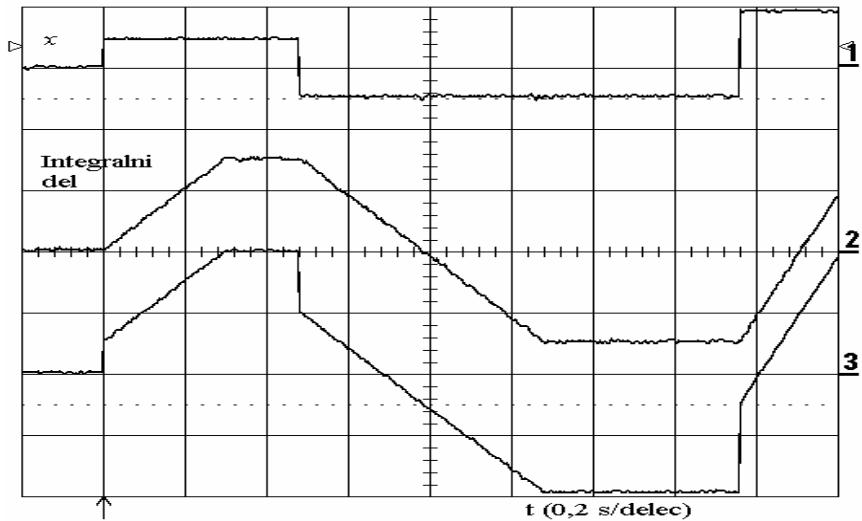
veliki vrednosti ali konstantnem predznaku vhoda) ali zaradi skupnega delovanja, to je njune vsote.

Potreba po omejevanju integralnega člena se pojavlja tudi ob nasičenju izvršnih elementov regulacijskega kroga (t.i. "wind-up"). Le-ti ne morejo vplivati na zmanjšanje regulacijskega pogreška, to je vhodne veličine PI regulatorja, zaradi česa bo integralni člen vztrajno povečeval izhodno vrednost. Poleg preprečevanja presežka je tukaj prisoten tudi problem vztrajnosti integratorja. Če npr. pride do nasičenja izhodnega signala iz PI regulatorja, se mora načeloma ustaviti tudi nadaljnje integriranje (slika 10. 28). V nasprotnem primeru bi I del še naprej integriral in bi pri hitri spremembi predznaka vhodnega signala na spremembo reagiral z zamudo, saj se je medtem "oddaljil" od delovno koristnega območja. Slike 10. 30 in 10. 31 kažeta blokovni shemi možnih realizacij PI člena. [30].



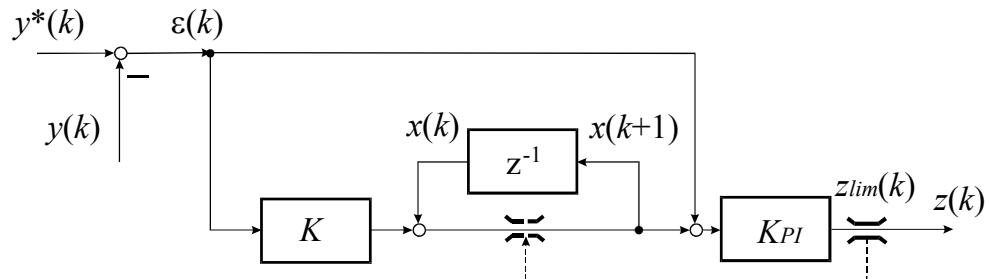
*Slika 10. 28: Limitiranje integralnega dela PI člena*

Oscilogram odziva regulatorja na različne stopničaste funkcije je na sliki 10. 29.

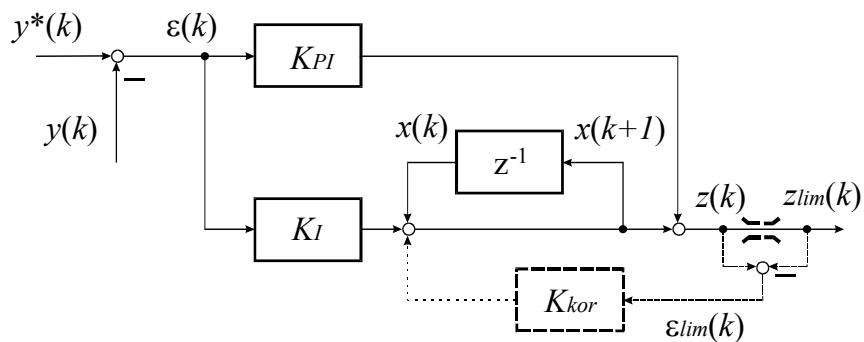


Slika 10. 29: Oscilogram odziva PI regulatorja ( $K_p = 1$ ,  $T_i = 100 \text{ ms}$ )

V prvem primeru realizacije PI člena na sliki 10. 30 imamo opravka z interakcijo med omejitvijo I člena ter celotnega PI člena ( $K = T_{ip}^{-1}$ , glej tudi sliko 10. 23). Na primeru s slike 10. 31 pa imamo opravka z dodatnim korekcijskim faktorjem, ki je aktivен ob nasičenju izhodne vrednosti PI člena.



Slika 10. 30: Blokovna shema možne rešitve PI člena (1)



Slika 10. 31: Blokovna shema možne rešitve PI člena (2)

### 10.3.5.7 Pulzno-širinska modulacija

Ojačenje izhodnega signala iz mikroračunalnika v elektromotorskih pogonih in podobnih vezjih močnostne elektronike je posebno poglavje, s katerim se tukaj ne bomo podrobno ukvarjali. V večini primerov uporabljamo tranzistorske ali tiristorske mostiče, ki delujejo v stikalnem režimu. Primer takega pretvorniškega vezja je prikazan na sliki 10. 5. Izmenično trifazno napetost poljubne oblike generiramo s programiranim spreminjanjem vklopnega časa posameznih tranzistorjev. V ta namen najpogosteje uporabljamo *pulzno-širinsko modulacijo* (angl. Pulse Width Modulation - PWM).

#### 10.3.5.7.1 Prostorski vektor

Napetost trifaznih simetričnih sistemov najlažje opišemo s *prostorskim vektorjem* (angl. space vector), ki je vsota prostorskih prispevkov vseh treh faz:

$$\vec{u}_S = \underbrace{u_{S1} \cdot e^{j0}}_{\vec{u}_{S1}} + \underbrace{u_{S2} \cdot e^{j\frac{2\pi}{3}}}_{\vec{u}_{S2}} + \underbrace{u_{S3} \cdot e^{j\frac{4\pi}{3}}}_{\vec{u}_{S3}}, \quad (10. 15)$$

kjer so  $u_{S1}$ ,  $u_{S2}$  in  $u_{S3}$  trenutne vrednosti faznih napetosti.

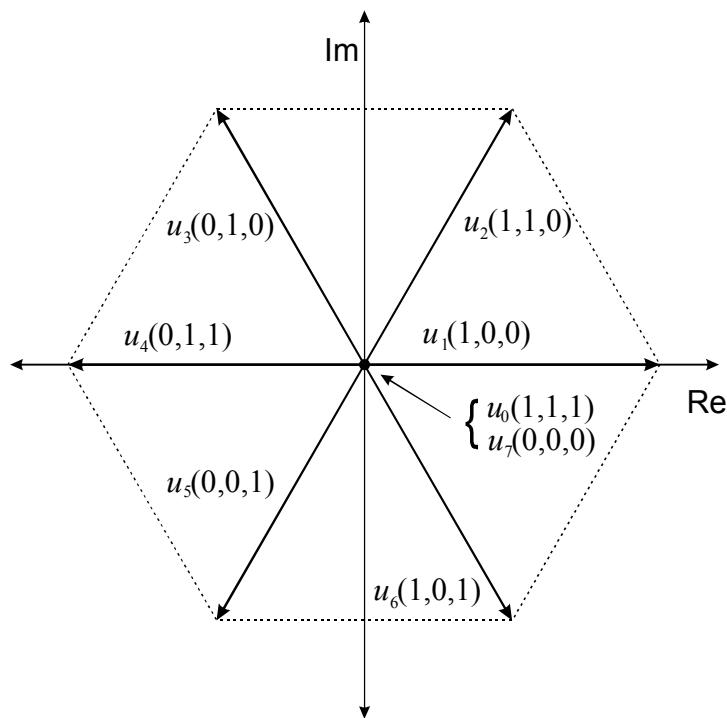
Zaradi diskretnega obratovanja imamo na voljo le omejeno število kombinacij, ki popisujejo stanje šestih tranzistorjev in faznih napetosti. Tranzistorji vklapljam ali izklapljajo enosmerno napetost vmesnega tokokroga na navitje ( $u_{vt}$  na sliki 10. 5). Seveda v nobenem primeru ne smemo hkrati vklopiti obe tranzistorjev v posamezni veji, saj bi to pomenilo kratek stik med sponkama  $u_{vt}$ . Pri vklopljenem zgornjem tranzistorju mora biti spodnji izklopljen, in obratno. Zato ponavadi prikazujamo le stanja zgornjih tranzistorjev, kjer "1" pomeni, da je tranzistor vklopljen, "0" pa, da je izklopljen. Število možnih kombinacij tranzitorskih stanj je  $2^3 = 8$ : 000, 001, 010, 011, 100, 101, 110 in 111.

Prvi bit se nanaša na stanje tranzistorjev S1 in S4, drugi na S2 in S5, tretji pa na S3 in S6. Splošni zapis prostorskih vektorjev se glasi (glej tudi sliko 10. 32)

$$\vec{u}_n = u_{vt} \cdot e^{j\frac{(n-1)\pi}{3}}, \quad \text{pri } n = 1 \dots 6$$

ter

$\vec{u}_0 = \vec{u}_7 = \vec{0}$ , ker je v tem primeru razlika potencialov med sponkami statorskih navitij enaka nič.

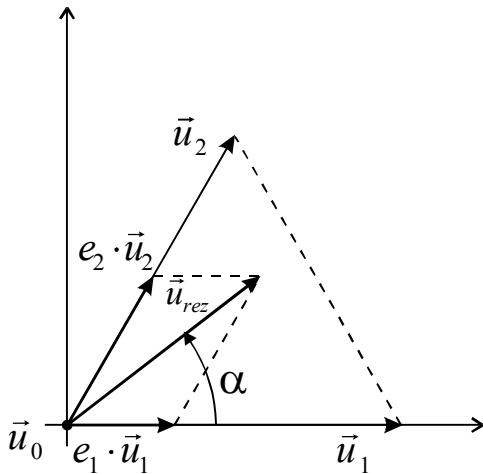


**Slika 10. 32: Prostorski vektor v odvisnosti od stanja stikal**

Osnovna naloga modulacije je generiranje poljubnega vektorja napetosti znotraj šestkotnika na sliki 10. 32 oz. tri fazne napetosti iz (10. 15) iz 6+2 vektorjev. To dosežemo s preklapljanjem med različnimi kombinacijami, kjer so sekvence prelopov in njihovo trajanje odvisne od izbranega načina modulacije. V tem poglavju bomo prikazali generiranje prostorskega vektorja napetosti prek ene od inačic pulzno-širinske modulacije.

#### 10.3.5.7.2 Naloga procesorja pri PWM

Kot primer si oglejmo stanje na sliki 10. 33. Treba je rešiti problem generiranja vektorja  $\vec{u}_{rez}$ . Vektor se nahaja v prvi šestini diagrama, torej je logično, da bosta pri njegovem oblikovanju sodelovala sosednja vektorja  $\vec{u}_1$  in  $\vec{u}_2$  (kombinacije stikal treh faz (1,0,0) in (1,1,0)), kakor tudi  $\vec{u}_0 = \vec{u}_7 = \vec{0}$  ((0,0,0) ali (1,1,1)).



**Slika 10. 33: Realizacija poljubnega vektorja napetosti**

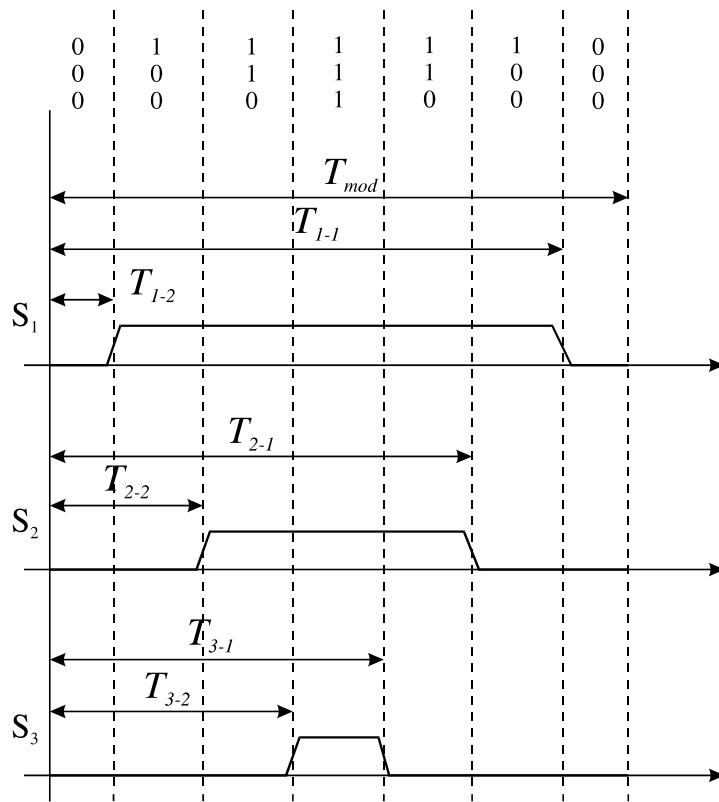
Poleg enosmerne napetosti vmesnega tokokroga  $u_{vt}$  je najpomembnejši parameter pri PWM frekvenca oz. interval modulacije  $T_{mod}$ . Znotraj tega intervala definiramo tudi časa vklopov  $T_1$  in  $T_2$  za vektorja  $\vec{u}_1$  in  $\vec{u}_2$ . Deleža posameznih vektorjev lahko definiramo s faktorjema  $e_1$  in  $e_2$ :

$$e_1 = \frac{T_1}{T_{mod}}, \quad e_2 = \frac{T_2}{T_{mod}}. \quad (10. 16)$$

Oba vektorja bosta znotraj intervala  $T_{mod}$  aktivno prispevala k oblikovanju povprečne vrednosti, to je rezultantnega vektorja  $\vec{u}_{rez}$ :

$$\vec{u}_{rez} = u_{rez} \cdot e^{j\alpha} = e_1 \cdot \vec{u}_1 + e_2 \cdot \vec{u}_2.$$

Na naslednji sliki je prikazan možen način proženja tranzistorjev za zgoraj opisani primer ter intervala vklopov obeh vektorjev. Takoj vidimo, da pri generiranju rezultante sodeluje tudi ničelni vektor. S spremenjanjem  $e_1$  in  $e_2$  oz.  $T_1$  in  $T_2$  lahko dosežemo poljuben vektor v trikotniku, ki je omejen z dvema kombinacijama stikal.



$$T_1 = (T_{2-2} - T_{1-2}) \times 2$$

$$T_2 = (T_{3-2} - T_{2-2}) \times 2$$

**Slika 10. 34: Sekvenca vklopov tranzistorjev v prvi šestini šestkotnika**

Sekvenca vklopov je naslednja:  $\vec{u}_0 \rightarrow \vec{u}_1 \rightarrow \vec{u}_2 \rightarrow \vec{u}_7 \rightarrow \vec{u}_2 \rightarrow \vec{u}_1 \rightarrow \vec{u}_0$  oz. kombinacije stikal  $(0,0,0) \rightarrow (1,0,0) \rightarrow (1,1,0) \rightarrow (1,1,1) \rightarrow (1,1,0) \rightarrow (1,0,0) \rightarrow (0,0,0)$ .

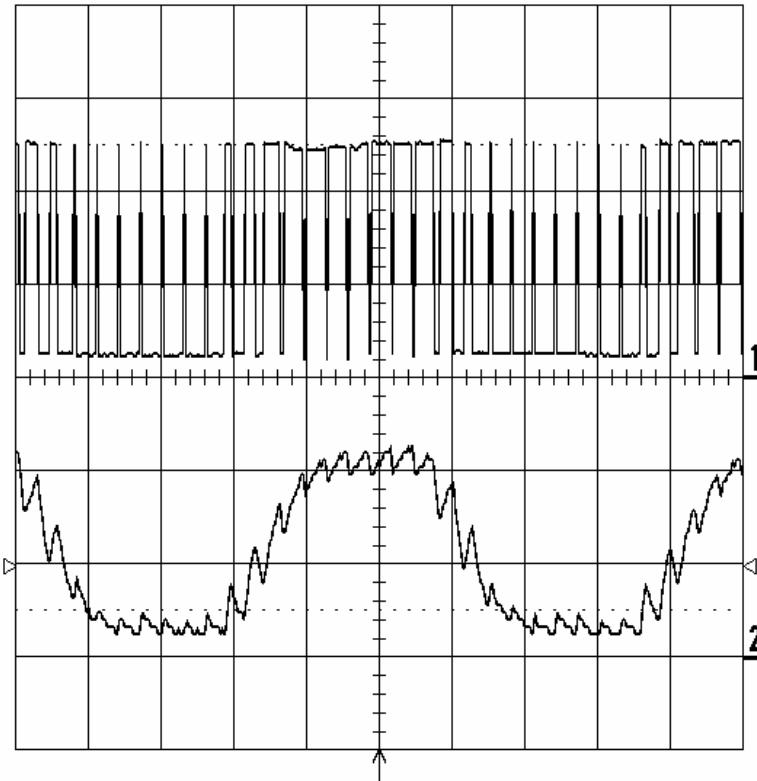
Mikroračunalnik mora pri generiranju signala za proženje tranzistorjev izvajati naslednje naloge:

1. Iz želenih napetosti izračunati velikost in smer želenega prostorskega vektorja. To pa hkrati pomeni detekcijo šestine, v kateri se ta vektor nahaja oz. kombinacije stikal dveh sosednjih vektorjev.
2. Izračunati trajanja vklopov posameznih vektorjev (en. (10. 16)).
3. Generirati vklopne sekvence s slike 10. 34.

Poleg tega je pri sodobnih mikrokontrolerjih (npr. TMS320F240) možna tudi *kompenzacija mrtvega časa*. Ob izračunih časov preklopov posameznih tranzistorjev namreč običajno predpostavljamo takojšnji vklop ali izklop, v praksi pa do tega pride z določeno zakasnitvijo. Posebej pri nizkih napetostih lahko ti mrtvi časi vplivajo na to, da dejanska napetost občutno nižja od želene in predpostavljeni.

Slika 10. 35 kaže oscilograma dejanske in filtrirane vrednosti krmilnega signala za enega od tranzistorjev mostiča ob uporabi PWM. V primeru je zaradi boljše vizualne ločljivosti pulzov

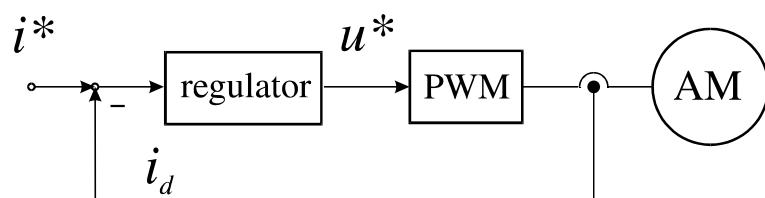
namenoma izbrana previsoka frekvenca želene sinusne napetosti (350 Hz), ki pri pogonih le redko pride v poštev. Zaradi tega je tudi filtrirana vrednost, ki naj bi ustrezala sinusu, nekoliko nazobčana. Frekvenca modulacije je 6,7 kHz (perioda 150  $\mu$ s).



**Slika 10. 35:** Oscilogram prožilnih pulzov enega tranzistorja kot posledica PWM ter njihova filtrirana vrednost (čas. baza 0,5 ms)

#### 10.3.5.7.3 Tokovni vir

Nekatere aplikacije (različni tokovni viri, izmenični motorji itd.) zahtevajo tokovno napajanje. Zato je pri sistemih s PWM potrebno zaključiti tokovno zanko, ki je v večini primerov skrajnja notranja zanka takšnih kaskadnih regulacij, s pa tem tudi najhitrejša (npr.  $T_V = 100\mu\text{s}$ ). Izvod iz regulatorja (običajno PI) toka je želena napetost, ki jo PWM blok mora realizirati.



**Slika 10. 36:** Tokovna zanka pri PWM

## 11. PRIMERI NEKATERIH MIKROKRMLNIKOV

V tem poglavju si bomo ogledali nekatere druge procesorske sisteme, ki jih pogosto srečujemo v napravah močnostne elektronike. Še posebej bomo govorili o digitalnih signalnih procesorjih, ki postajajo vse zanimivejši tudi pri cenejših aplikacijah. Tabela 11.1 kaže kratek zgodovinski razvoj nekaterih pomembnejših mikroprocesorjev in mikrokrmlnikov [5].

| Procesor      | Predstavitev | Pod. vodilo | Ura     | Nasl. vodilo | Značilnosti          |
|---------------|--------------|-------------|---------|--------------|----------------------|
| Intel 8080    | 1975         | 8 bit       |         | 16 bit       |                      |
| MC6800        | 1976         | 8 bit       | 2 MHz   | 16 bit       |                      |
| Intel 8086    | 1978         | 16 bit      | 8 MHz   | 20 bit       | FPCP, DNR            |
| MC6805        | 1979         | 8 bit       | 2 MHz   | 16 bit       | CNTRL                |
| MC68000       | 1981         | 16 bit      | 8 MHz   | 23 bit       | FPCP, DNR            |
| MC68HC11      | 1984         | 8 bit       | 4 MHz   | 16 bit       | CNTLR                |
| MC68040       | 1989         | 32 bit      | 25 MHz  | 30 bit       | C, FPPL, IPL,<br>DNR |
| Intel 486     | 1991         | 32 bit      | 66 MHz  | 30 bit       | C, FP, IPL, DNR      |
| MC68332       | 1991         | 16 bit      | 17 MHz  | 24 bit       | CNTLR                |
| DEC alpha     | 1992         | 64 bit      | 200 MHz | 34 bit       | C, FPPL, IPL         |
| SuperSparc    | 1992         | 32 bit      | 100 MHz | 32 bit       | C, FPPL, IPL         |
| MPC601        | 1993         | 64 bit      | 80 MHz  | 32 bit       | C, FPPL, IPL         |
| Intel Pentium | 1993         | 32 bit      | 66 MHz  | 30 bit       | C, FPPL, IPL         |

**Legenda:**

FPCP = obstaja tudi matematični koprocesor, CNTLR = mikrokrmlnik, C = cache pomnilnik na čipu, FP = interna plavajoča vejica, IPL = integer pipeline, FPPL = pipeline s plavajočo vejico

**Pojasnilo:**

*Cache* (slov. predpomnilnik): zelo hiter pomnilnik z ažurirano kopijo zadnjih uporabljenih besed v glavnem pomnilniku.

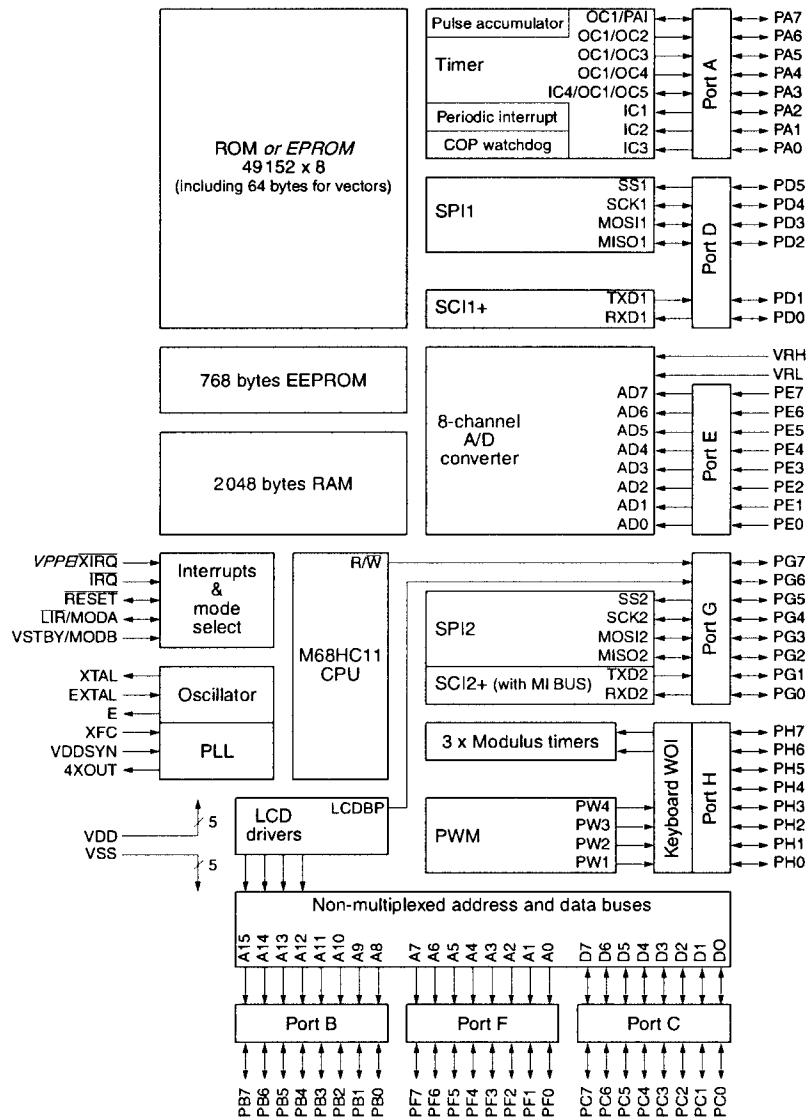
*Pipeline* (slov. cevovod): Sistem funkcionalnih enot za sočasno izvajanje ukazov v različnih fazah.

**Tabela 11.1: Kratek zgodovinski pregled mikroprocesorjev in mikrokrmlnikov**

### 11.1 Klasični mikrokrmlniki

#### 11.1.1 Motorola MC68HC711PH8

Motorolina serija HC11 je že veliko let ena najbolj posrečenih rešitev za procesorsko manj zahtevne naloge. CPU izvira iz procesorja 6800, zato je njegov nabor ukazov nekompatibilen z ukazi MC68332, čeprav imata oba čipa podobno krmilniško strukturo.

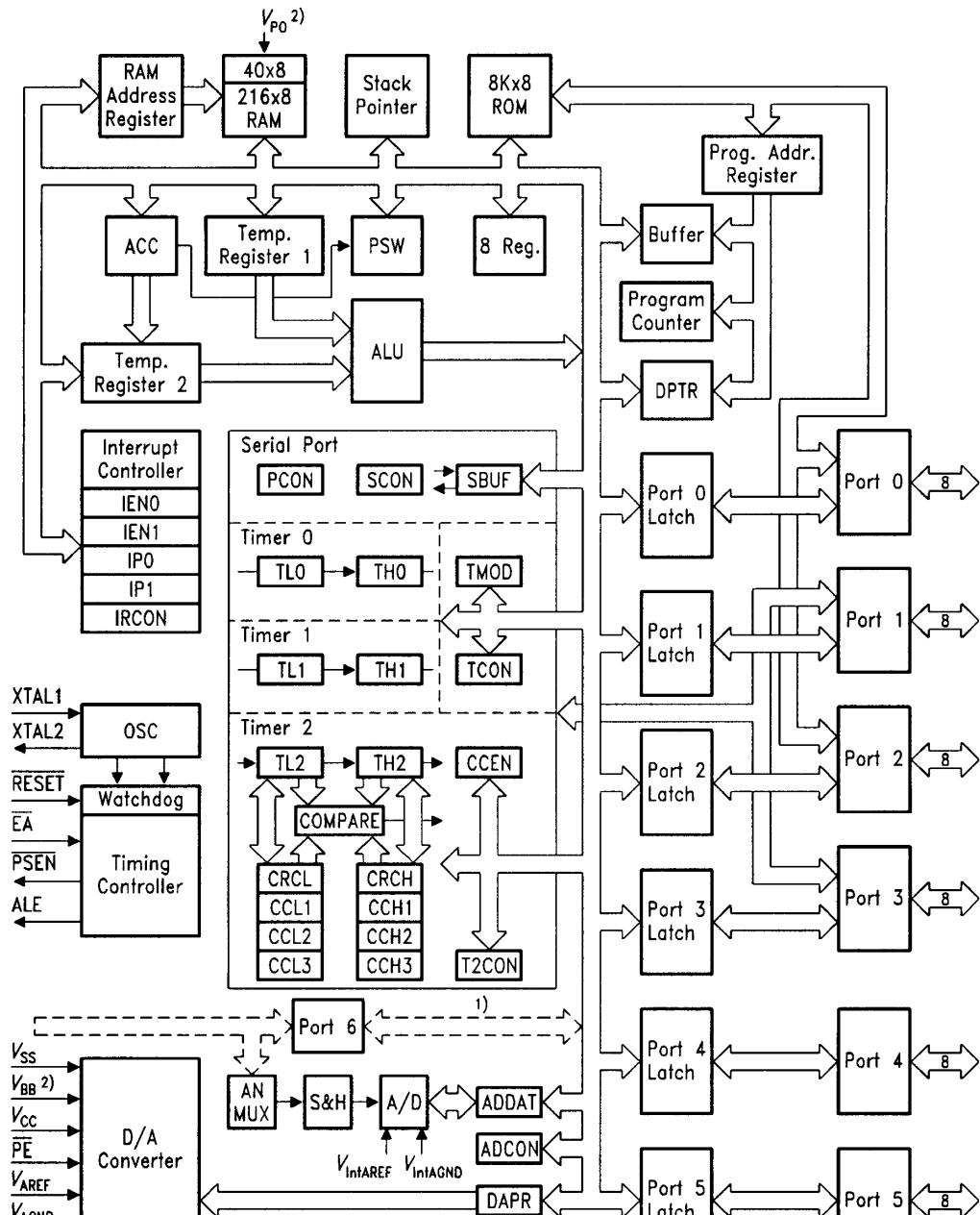


Slika 11.1: Blokovna shema mikrokrmlnika MC68HC711PH8

### Značilnosti:

- PLCC ohišje (angl. plastic-leaded chip carrier),
- ura 2 - 4 MHz,
- 48 Kbyte EPROM na čipu,
- 2 Kbyte RAM pomnilnika,
- 768 byte EEPROM,
- do 54 binarnih vhodno/izhodnih nožic za splošne namene,
- 16-bitni časovnik,
- trije 8-bitni časovniki za generiranje periodičnih prekinitvev,
- dva 8- ali 9-bitna podsistema za asinhronsko serijsko komunikacijo (SCI),
- 8 kanalni 8-bitni A/D pretvornik,
- štirje 8-bitni kanali za pulzno-širinsko modulacijo (PWM),
- 4-segmentni goničnik za prikazovalnik s tekočimi kristali (angl. LCD driver).

### 11.1.2 Siemens SAB 80(C)535



- 1) Dodatna značilnost ACMOS verzije  
 2) Dodatna značilnost MYMOS verzije

Slika 11.2: Blokovna shema mikrokrmlnika 80C535

SAB 80(C)535 in njegova različica 80(C)515 sta pripadniki zelo uspešne Siemensove družine 8-bitnih mikrokrmlnikov.

#### Značilnosti:

- 8 Kbyte programskega pomnilnika na čipu,

- 256 byte RAM-a na čipu,
- 6 paralelnih 8-bitnih vhodno/izhodnih portov,
- trije 16-bitni časovniki/števci,
- A/D pretvornik z osmimi multipleksiranimi vhodi, programirljiva referenčna napetost,
- 12 virov prekinitev (7 notranjih, 5 zunanjih),
- globina sklada do 256 bytov,
- čas izvajanja ukazov 1 µs (pri 12 MHz uri),
- čas izvajanja operacije deljenja 4 µs.

## 11.2 RISC procesorji in mikrokrmlniki

Za klasične mikroprocesorje (angl. CISC - Complex Instruction Set Computer) je značilno veliko število različnih ukazov ter načinov naslavljanja operandov, kar smo opisali v poglavju 4 o CPU-ju MC68332. To seveda zahteva zapleteno strukturo procesorja in kompleksne sekvence izvajanja ukazov. Nekateri ukazi so sestavljeni iz več osnovnih besed procesorja (mikrokoda).

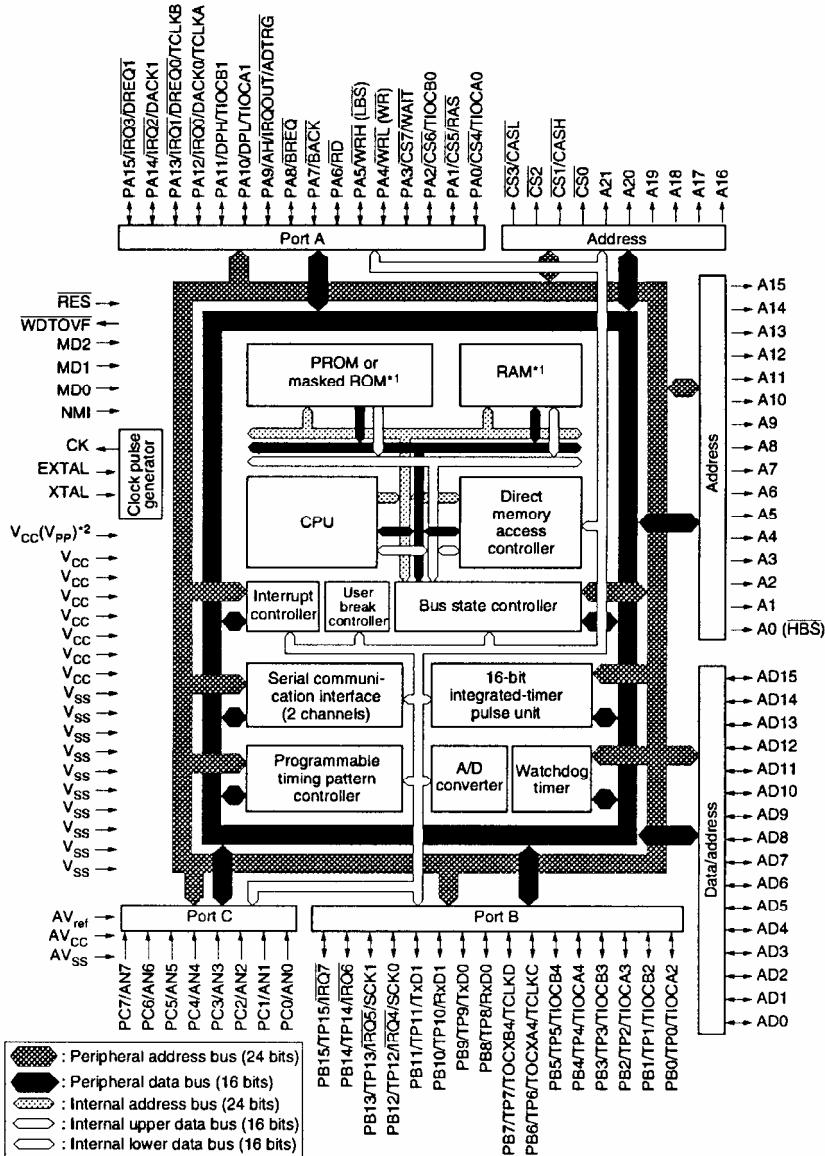
Iz statistične analize uporabe ukazov ugotovimo, da so najpogosteje uporabljeni ukazi za prenos podatka, seštevanje, pogojne skoke in primerjave. Po drugi strani se nekateri ukazi zelo redko uporabljajo. Ta ugotovitev je pripomogla k nastajanju *RISC* arhitekture (angl. Reduced Instruction Set Computer - računalniki z zmanjšanim naborom ukazov). Z zmanjšanim številom ukazov in načinov naslavljanja je omogočeno izvajanje večine ukazov v enem časovnem ciklu. Hkrati je tudi potreba po mikrokodiranju postala odvečna, ker je vlogo mikrokodiranja ukazov prevzel hardver procesorja. Zaradi hitrosti vsebujejo RISC računalniki zelo hitre pomnilniške enote (npr. hitre statične RAM pomnilnike - SRAM za podatkovni in inštrukcijski cache). Več o RISC arhitekturi lahko preberemo v literaturi [npr. 22].

Poleg v zelo zmogljivih računalnikih, se je RISC začel uveljavljati tudi v izvebah, ki so namenjene procesni regulaciji. Primera takšnih procesorjev sta Hitachijeva RISC-a SH7032/SH7034.

Nekatere lastnosti SH7032/SH7034:

- frekvenca 20 MHz pri 5 V ali 12,5 pri 3,3 V,
- 0,8 mikronska tehnologija,
- 8 Kbyte RAM (SH7032) ali 64 Kbyte ROM/EPROM in 4 Kbyte RAM (SH7034),
- CPU: super RISC engine,
- 32-bitno notranje vodilo,
- petstopenjski cevovod (pipeline),
- operacije množenja  $32 \times 32 = 64$  se izvajajo v dveh ali treh ciklih,
- MAC (množenje in seštevanje) se izvrši v dveh ali treh ciklih,
- štirikanalni DMA krmilnik,
- 9 zunanjih prekinitvenih virov,
- PWM modul,
- števci, zajemanje in merjenje vhodnih pulzov,
- 8-kanalni 10-bitni A/D pretvornik s spremenljivo referenčno napetostjo,
- dvokanalni SCI modul,

- 32 vhodno/izhodnih nožic in dodatnih 8 vhodnih nožic.



Slika 11.3: Blokovna shema RISC krmilnika Hitachi SH7032/7034

### 11.3 Digitalni signalni procesorji in mikrokrmlniki

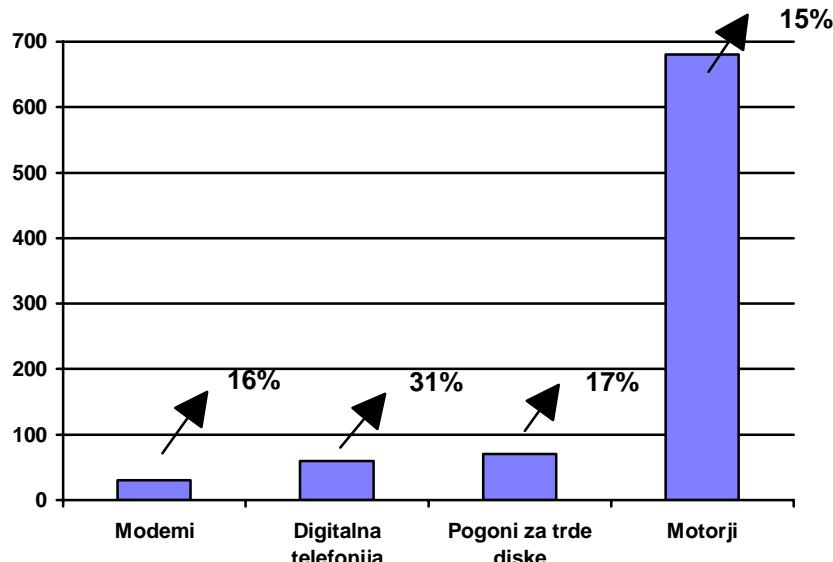
Digitalni signalni procesorji (Digital Signal Processors - DSP) so bili prvotno namenjeni digitalni obdelavi analognih signalov (npr. zvoka ali slike), kjer obstaja potreba po zelo hitrem računanju v realnem času (npr. hitra Fourierjeva transformacija - FFT). Čeprav imajo DSP-ji v primerjavi s klasičnimi procesorji enako stopnjo integracije in frekvenco urinega takta, so primernejši za takšne naloge zaradi optimizirane arhitekture<sup>1</sup>.

<sup>1</sup> Primerjava: starejši klasični procesor Intel 80386 porabi za FFT z N=1024 300 ms, enako star DSP TMS320C30 pa 1,5 ms [14].

Kratka primerjava DSP-jev in klasičnih procesorjev:

1. **Aritmetična enota:** DSP-ji so optimirani za izvajanje operacij množenja in hkratnega seštevanja (npr.  $\Sigma a_i b_i$ ). Zato je množilna enota sestavni del DSP-ja in omogoča izvajanje kombiniranega množenja in seštevanja v enem samem urinem ciklu! Po drugi strani pa porabijo klasični procesorji za izvajanje množenja veliko časovnih ciklov (običajno nekajdeset). Poleg tega vsebujejo DSP-ji tudi posebno enoto za operacije pomika (shift).
2. **Arhitektura vodila:** Običajni procesorji so zasnovani na von Neumanovi arhitekturi vodil (pogl. 2): skupno podatkovno in ukazno pomnilniško področje ter po eno podatkovno in naslovno vodilo. Zato je izvajanje ukaza sestavljeno iz nekaj korakov: branje ukaza iz pomnilnika, dekodiranje, branje operanda in izvajanje operacije ukaza, saj podatki in ukazi "potujejo" po istem vodilu. DSP-ji uporabljajo Hardvard arhitekturo z ločenima vodiloma za ukaze in podatke (angl. program bus, data bus). Na ta način je omogočeno hkratno branje obeh (pri nekaterih lahko dodatno preberemo hkrati dva podatka oz. operanda - t.i. modifcirana Harvard arhitekturo).
3. **Naslavljanje:** Klasični mikroprocesorji omogočajo veliko načinov naslavljanja, med njimi tudi takšne, ki so prirejeni za obdelave velikih podatkovnih polj. Pri tem je čas, potreben za izračun efektivnega naslova običajno daljši od same operacije, ki jo ukaz izvaja. DSP-ji vsebujejo posebne *generatorje naslova* (angl. address generators), ki omogočajo bistveno hitrejše naslavljjanje.
4. **Pomnilnik:** mikroprocesorji shranjujejo ukaze in podatke v zunanjem pomnilniku (šele mikrokrmilniki vsebujejo nekaj spomina na čipu). V prejšnjih poglavjih smo že povedali, da je takšno delo z zunanjim pomnilnikom počasnejše od uporabe internega pomnilnika na čipu, ki so sestavni del DSP-jev.

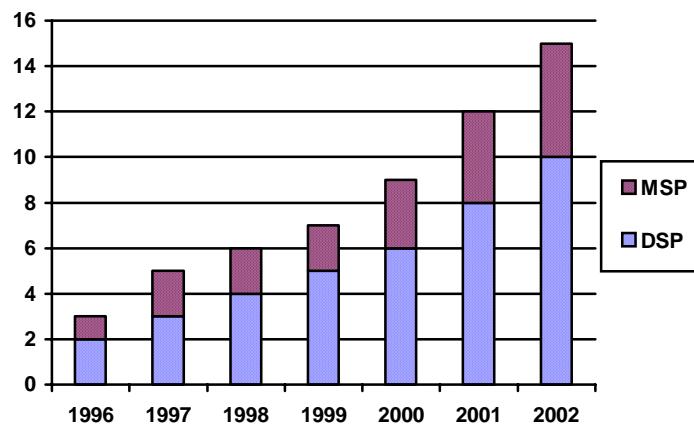
Kljub naštetim prednostim sta bila do nedavnega krmiljenje in regulacija vezij močnostne elektronike rezervirana izključno za klasične procesorje. Razlog za to je bila visoka cena DSP-jev. Ta trend se spreminja, saj proizvajalci danes ponujajo posebne DSP krmilnike, ki so optimirani za takšne naloge, po izredno nizkih cenah (cena posameznega kosa serije TMS320xx je okrog 12 dolarjev, nad 10000 kosov pa okrog treh dolarjev). Zato DSP-ji prodirajo tudi na to področje (slika 11.4).



Vir: Dataquest, Motion Tech Trends

**Slika 11.4: Število prodanih DSP-jev (v milijonih) leta 1996 po področjih ter trendi rasti**

Trenutno je največji proizvajalec DSP-jev Texas Instruments s 45-odstotnim tržnim deležem (Lucent: 29 %, ADI: 11 %, Motorola 8 %). Tržni delež DSP-jev v primerjavi z ostalimi polprevodniškimi komponentami (MSP-Mixed Signal Products) zgovorno kaže slika 11.5. Vrednost trga s DSP-ji raste 50 % hitreje od trga z ostalimi integriranimi vezji [vir: TMS320 DSP Solutions, TI, 1997, CD].

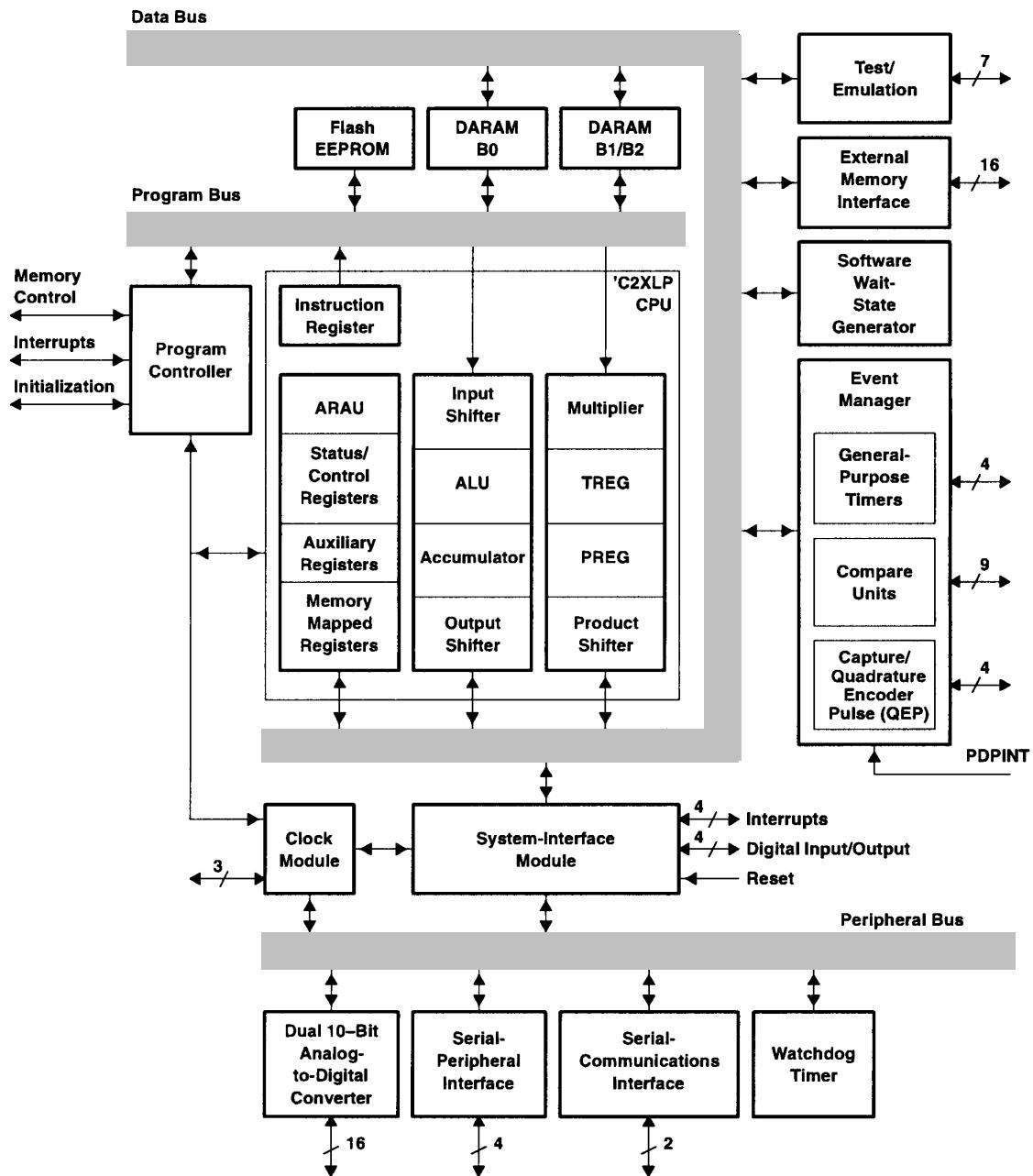


**Slika 11.5: Stanje in projekcija tržnega deleža DSP-jev in MSP firme Texas Instruments v milijardah dolarjev**

### 11.3.1 Digitalni signalni mikrokrmlnik TMS320F240

DSP mikrokrmlnik Texas Instruments TMS320F240 pomeni prelomnico na področju mikroprocesorskih komponent za aplikacije v močnostni elektroniki in električnih pogonih. DSP jedro je zaslužno za izredno hitrost izvajanja ukazov, ki so primerni za zahtevne

matematične operacije. Njegov periferni del je optimiran za krmiljenje in regulacijo električnih motorjev. Mikrokrmilnik vsebuje tudi precej programskega in podatkovnega pomnilnika in ga v večini primerov lahko samostojno uporabimo. Kljub veliki zmogljivosti je proizvajalcu uspelo zagotoviti izjemno nizko ceno, ki se giblje od 4 do 18 dolarjev za kos. Zato je mikrokrmilnik zelo zanimiv tudi pri manjših pogonih (npr. pralnih strojih, klimatskih napravah itd.). TMS320x240 že diktira razvojne tendre na tem področju, zato si ga bomo ogledali nekoliko podrobneje [28, 29, 30, 32]. Njegovo blokovno shemo kaže slika 11.6.



Slika 11.6: Blokovna shema DSP krmilnika TMS32F0240

Osnovne značilnosti:

- CMOS tehnologija,
- čas izvajanja ukaznega cikla 50 ns,
- 16-bitna procesorska enota s fiksno vejico (angl. fixed point),
- ukazi z istočasnim branjem dveh ali treh operandov,
- Pomnilnik: 544 besed x 16 bitov RAM-a z dvojnim dostopom na čipu (Dual-Access RAM - DARAM), 16 K besed x 16 bitov FLASH eprom-a (inačica "F240") ali ROM-a (inačica "C240"), celotno naslovljivo pomnilniško področje 224 K besed x 16 bitov,
- arhitektura z več vodili (multibus): ločena programska, podatkovna in periferna vodila,
- 12 kanalov s primerjalnimi ali PWM funkcijami (glej poglavje 7 o TPU MC68332),
- trije neodvisni časovniki,
- dvojni 10 bitni A/D modul s časom pretvorbe 10 µs,
- 28 neodvisno programirljivih multipleksiranih vhodno/izhodnih nožic,
- modul za serijsko komunikacijo SCI (glej poglavje 6.3 o SCI MC68332),
- periferni serijski vmesnik SPI (glej poglavje 6.2 o QSPI MC68332),
- enota za nadzorovanje dogodkov (podobno TPU enoti MC68332),
- 6 zunanjih prekinitev.

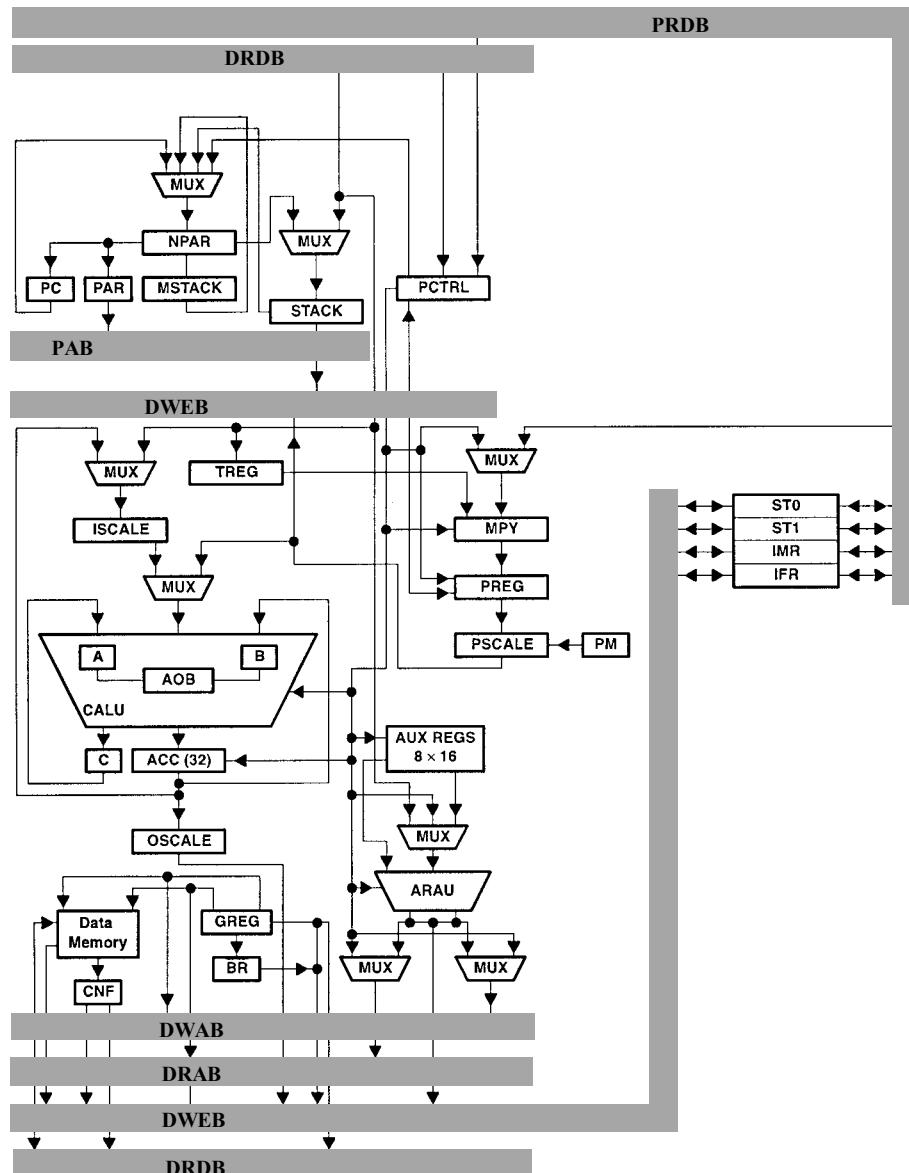
#### 11.3.1.1 Struktura TMS320F240

Ena od lastnosti DSP-jev sta ločeni pomnilniški področji za ukaze (angl. program space) in podatke (angl. data space) ter temu ustrezno ločena vodila. Na ta način je možna hkratna uporaba obeh področij. Poleg tega ima mikrokrmilnik še posebno vodilo za naslavljjanje periferije (angl. peripheral bus) na katerega sta vezana interna A/D pretvornika, enota za serijsko komunikacijo, serijski periferni vmesnik itd. (slika 11.6). Slika 11.7 kaže strukturo CPU-ja.

Osnovne komponente CPU-ja so:

1. 32-bitna **centralna aritmetično-logična enota** (CALU) omogoča izvajanje širokega spektra aritmetičnih in logičnih operacij, večinoma v enem urinem ciklu (50 ns).
2. 32-bitni **akumulator** vsebuje rezultat operacije iz CALU.
3. **Množilna enota** ( $16 \times 16 = 32$ ) omogoča izvajanje operacij predznačenega ali nepredznačenega množenja v enem ciklu. Vhodni podatek je shranjen v 16-bitnem registru TREG (drugi operand je vsebovan v samem ukazu), rezultat pa se nahaja v 32-bitnem PREG registru.
4. 16-bitni **skalirni pomicni registri** omogočajo pomik podatkov v/iz ALU in množilne enote, kar je koristno pri prilagajanju vhodnih podatkov in rezultata (npr. v formatu f4.14).
5. **Pomožni registri** (AR0-AR7) so zelo koristni pri posrednem naslavljjanju, kjer s **pomožno aritmetično enoto** (ARAU) hitro izračunamo efektivni naslov.
6. **Ostali registri:** statusna regista (ST1 in ST0), register za maskiranje prekinitev (IMR) in register za indikacijo prekinitev (IFR).

Organizacija CPU-ja omogoča tudi hkratno izvajanje več operacij (npr. **MPYA**: množenje in prštevanje prejšnjega produkta k vsebini akumulatorja ob hkratnem premiku).



Slika 11.7: Blokovni diagram CPU-ja pri mikrokrmlniku TMS320F240

### 11.3.1.2 Organizacija pomnilnika

Mikrokrmlnik lahko deluje v dveh režimih: mikrokrmlniškem (nožica  $MP/MC = "0"$ ) in mikroprocesorskem ( $MP/MC = 1$ ). Notranji Flash EEPROM (pogl. 12.1.3) je dostopen le v prvem režimu. V ta pomnilnik shranimo končno verzijo programa, po potrebi pa ga tudi spremenimo. Na sliki 11.8 sta prikazani pomnilniški mapi programskega področja za oba primera. Uporaba DARAM-a je možna brez čakalnih stanj, njegov položaj v programskem ali podatkovnem področju pa je odvisen od stanja bita CNF v statusnem registru ST1.

| <b>Program Space</b><br><b>MP/MC=1</b><br><b>Microprocessor Mode</b> |  | <b>Program Space</b><br><b>MP/MC=0</b><br><b>Microcomputer Mode</b> |   |
|--|--|---|---|
| <b>Hex</b>   | <b>Interrupts</b><br><b>(External)</b> | <b>Hex</b>  | <b>Interrupts</b><br><b>(On-Chip)</b>   |
| 0000   |  | 0000  |   |
| 003F   |  | 003F  |   |
| 0040   | <b>External</b>                        | 0040  |   |
| FDFD   |  | 3FFF  | On-Chip ROM<br>(Flash EEPROM)<br>(8 x 2K Segments)<br>(Seg 0 = Boot Seg<br>@ 0h-07FFh by<br>BOOTPROT pin) |
| FE00   | On-Chip DARAM<br>B0 (CNF = 1)          | 4000  | <b>External</b>   |
| FEFF   | <b>External (CNF = 0)</b>              | FE00  | On-Chip DARAM<br>B0 (CNF = 1)   |
| FF00   | On-Chip DARAM<br>B0' (CNF = 1)         | FEFF  | <b>External (CNF = 0)</b>   |
| FFFF   | <b>External (CNF = 0)</b>              | FF00  | On-Chip DARAM<br>B0' (CNF = 1)  |
|  |  | FFFF  | <b>External (CNF = 0)</b>   |

*Slika 11.8: Pomnilniška mapa programskega področja*

Mapo podatkovnega področja kaže slika 11.9. Področje od 7000<sub>HEX</sub> do 743f<sub>HEX</sub> je rezervirano za registre podmodulov mikrokrmlnika (podobno kot pri MC68332), s katerimi izberemo načine njihovega delovanja.

Tukaj velja omeniti še eno posebnost naslavljanja podatkovnega področja v DSP-ju. Celotno področje je sestavljeno iz 512 podatkovnih strani (angl. data pages - DP) s 128 besedami (slika 11.10). Naslov podatka je sestavljen iz dveh delov:

- Kazalec na naslov strani se nahaja v 9-bitnem področju ST0 ( $2^9 = 512$ ). Naslov strani spremenimo z ukazom LDP (angl. load data page pointer).
- Naslov znotraj podatkovne strani je sestavljen iz sedmih bitov ( $16 - 9 = 7 \rightarrow 2^7 = 128$ ), ki so del vsakega ukaza za naslavljjanje podatkov (angl. offset).

Omenjeni način omogoča bistveno krajše naslavljjanje, programer pa mora vedno skrbeti za to, da se nahaja na "pravi" strani. V nasprotnem primeru isti ukaz z relativnim naslovom lahko naslovi podatek na neki drugi strani.

| Hex  |   |
|------|---|
| 0000 | Memory-Mapped Registers and Reserved  |
| 005F |   |
| 0060 | On-Chip DARAM B2  |
| 007F |   |
| 0080 | Reserved  |
| 00FF |   |
| 0100 | On-Chip DARAM B0 (CNF = 0)  |
| 01FF | Reserved (CNF = 1)  |
| 0200 | On-Chip DARAM B0' (CNF = 0)   |
| 02FF | Reserved (CNF = 1)  |
| 0300 | On-Chip   |
| 03FF | DARAM B1  |
| 0400 | On-chip   |
| 04FF | DARAM B1'   |
| 0500 |   |
| 07FF | Reserved  |
| 0800 |   |
| 6FFF | Illegal   |
| 7000 | Peripheral Memory-Mapped Registers (System, ADC, SCI, SPI, I/O, Interrupts) |
| 73FF |   |
| 7400 | Peripheral Memory-Mapped Registers (Event Manager)                          |
| 743F |   |
| 7440 | Reserved  |
| 77FF |   |
| 7800 | Illegal   |
| 7FFF |   |
| 8000 | External  |
| FFFF |   |

Slika 11.9: Pomnilniška mapa podatkovnega področja

| Stanje DP   | Preostali del naslova (odmik) | Podatkovni pomnilnik                                   |
|-------------|-------------------------------|--|
| 0000 0000 0 | 000 0000                      | Stran 0: 0000 <sub>HEX</sub> - 007f <sub>HEX</sub>     |
| 0000 0000 0 | 111 1111                      |  |
| 0000 0000 1 | 000 0000                      | Stran 1: 0080 <sub>HEX</sub> - 00ff <sub>HEX</sub>     |
| 0000 0000 1 | 111 1111                      |  |
| 0000 0001 0 | 000 0000                      | Stran 2: 0100 <sub>HEX</sub> - 017f <sub>HEX</sub>     |
| 0000 0001 0 | 111 1111                      |  |
| .           | .                             | .  |
| 1111 1111 1 | 000 0000                      | Stran 511: ff80 <sub>HEX</sub> - ffffff <sub>HEX</sub> |
| 1111 1111 1 | 111 1111                      |  |

Slika 11.10: Strani podatkovnega pomnilnika

Tretje pomnilniško področje je namenjeno delu s perifernimi enotami. V njem se nahajajo tudi registri, potrebeni za programiranje Flash EPROM-a.

#### 11.3.1.3 Periferija

TMS320F240 vsebuje naslednje periferne enote:

1. vmesnik k zunanjim pomnilnikom,
2. časovna straža (watchdog),
3. enoto za serijsko komunikacijo,
4. serijski periferni vmesnik,
5. dvojni A/D pretvornik,
6. nadzorovanje dogodkov (angl. event manager)

Naštete enote smo spoznali že pri MC68332. Tukaj bomo omenili le dve.

##### 11.3.1.3.1 A/D pretvornik

A/D modul sestoji iz dveh 10-bitnih pretvornikov. Oba pretvornika lahko izbirata med osmimi multipleksiranimi kanali napetostnega območja 0 V - 5 V. Pretvorba traja 10 µs, kar pa ne pomeni, da je v tem času onemogočena obdelava ostalih ukazov v programu. Interna struktura po zagonu omogoča shranjevanje rezultata v posebna FIFO regista, ki vsebuje veljavni podatek po zaključku pretvorbe. CPU lahko vsebinsi registrov prebere v kateremkoli trenutku. Vzorčeni podatek je veljaven le zagotoviti, če je med zagonom pretvorbe in branjem pretekel deklarirani čas pretvorbe.

##### 11.3.1.3.2 Enota za nadzorovanje dogodkov

Enota za nadzorovanje dogodkov (angl. event manager - EV) je podobna TPU enoti MC68332, saj se ukvarja z zajemanjem ali generiranjem binarnih signalov nivoja 0 V - 5 V. Glavni sestavni deli so:

1. časovniki za splošno rabo,
2. primerjalna enota,
3. enota za zajemanje signalov,
4. enota za zajemanje signalov iz inkrementalnega dajalnika (QEP).

Ena od nalog **primerjalne enote** (angl. compare unit) je generiranje pulzno-širinsko moduliranega signala na devetih izhodih. Te signale lahko uporabljam za proženje tranzistorjev v močnostnem vezju. Ena od posebnosti PWM obratovanja pri TMS320F240 je možnost kompenzacije mrtvega časa preklopov.

#### 11.3.1.4 Nabor ukazov

Naslednje tabele kažejo celoten nabor ukazov TMS320F240. Ukazov ne bomo posebej obravnavali, bralec pa jih lahko sam primerja z ukazi klasičnih mikroprocesorjev (npr. CPU

MC68332). Stolpec "Besede" kaže dolžino posameznega ukaza, stolpec "Cikli" pa njegovo trajanje (mnogokratnik 50 ns).

| MNEMONIK | OPIS   | BESEDE | CIKLI |
|----------|--|--------|-------|
| ABS      | Absolute value of ACC  | 1      | 1     |
| ADD      | Add to ACC with shift of 0 to 15, direct or indirect                         | 1      | 1     |
|          | Add to ACC with shift 0 to 15, long immediate                                | 2      | 2     |
|          | Add to ACC with shift of 16, direct or indirect                              | 1      | 1     |
|          | Add to ACC, short immediate  | 1      | 1     |
| ADDC     | Add to ACC with carry, direct or indirect                                    | 1      | 1     |
| ADDS     | Add to low ACC with sign-extension suppressed, direct or indirect            | 1      | 1     |
| ADDT     | Add to ACC with shift (0 to 15) specified by TREG, direct or indirect        | 1      | 1     |
| AND      | AND ACC with data value, direct or indirect                                  | 1      | 1     |
|          | AND with ACC with shift of 0 to 15, long immediate                           | 2      | 2     |
|          | AND with ACC with shift of 16, long immediate                                | 1      | 2     |
| CMPL     | Complement ACC   | 1      | 1     |
| LACC     | Load ACC with shift of 0 to 15, direct or indirect                           | 1      | 1     |
|          | Load ACC with shift of 0 to 15, long immediate                               | 2      | 2     |
|          | Load ACC with shift of 16, direct or indirect                                | 1      | 1     |
| LACL     | Load low word of ACC, direct or indirect                                     | 1      | 1     |
|          | Load low word of ACC, short immediate  | 1      | 1     |
| LACT     | Load ACC with shift (0 to 15) specified by TREG, direct or indirect          | 1      | 1     |
| NEG      | Negate ACC   | 1      | 1     |
| NORM     | Normalize the contents of ACC, indirect                                      | 1      | 1     |
| OR       | OR ACC with data value, direct or indirect                                   | 1      | 1     |
|          | OR with ACC with shift of 0 to 15, long immediate                            | 2      | 2     |
|          | OR with ACC with shift of 16, long immediate                                 | 2      | 2     |
| ROL      | Rotate ACC left  | 1      | 1     |
| ROR      | Rotate ACC right   | 1      | 1     |
| SACH     | Store high ACC with shift of 0 to 7, direct or indirect                      | 1      | 1     |
| SACL     | Store low ACC with shift of 0 to 7, direct or indirect                       | 1      | 1     |
| SFL      | Shift ACC left   | 1      | 1     |
| SFR      | Shift ACC right  | 1      | 1     |
| SUB      | Subtract from ACC with shift of 0 to 15, direct or indirect                  | 1      | 1     |
|          | Subtract from ACC with shift of 0 to 15, long immediate                      | 2      | 2     |
|          | Subtract from ACC with shift of 16, direct or indirect                       | 1      | 1     |
|          | Subtract from ACC, short immediate   | 1      | 1     |
| SUBB     | Subtract from ACC with borrow, direct or indirect                            | 1      | 1     |
| SUBC     | Conditional subtract, direct or indirect                                     | 1      | 1     |
| SUBS     | Subtract from ACC with sign-extension suppressed, direct or indirect         | 1      | 1     |
| SUBT     | Subtract from ACC with shift (0 to 15) specified by TREG, direct or indirect | 1      | 1     |
| XOR      | Exclusive OR ACC with data value, direct or indirect                         | 1      | 1     |
|          | Exclusive OR with ACC with shift of 0 to 15, long immediate                  | 2      | 2     |
|          | Exclusive OR with ACC with shift of 16, long immediate                       | 2      | 2     |
| ZALR     | Zero low ACC and load high ACC with rounding, direct or indirect             | 1      | 1     |

**Tabela 11.2: Akumulatorski, aritmetični in logični ukazi**

| MNEMONIK | OPIS   | BESEDE | CIKLI |
|----------|--|--------|-------|
| ADRK     | Add constant to current AR, short immediate                                | 1      | 1     |
| BANZ     | Branch on current AR not 0, indirect                                       | 2      | 4     |
| CMPR     | Compare current AR with AR0  | 1      | 1     |
| LAR      | Load specified AR from specified data location, direct or indirect         | 1      | 2     |
|          | Load specified AR with constant, short immediate                           | 1      | 2     |
|          | Load specified AR with constant, long immediate                            | 2      | 2     |
| MAR      | Modify current AR and/or ARP, indirect (performs no operation when direct) | 1      | 1     |
| SAR      | Store specified AR to specified data location, direct or indirect          | 1      | 1     |
| SBRK     | Subtract constant from current AR, short immediate                         | 1      | 1     |

**Tabela 11.3: Ukazi pomožnih (angl. auxiliary) registrov**

| MNEMONIK | OPIS  | BESEDE | CIKLI |
|----------|---|--------|-------|
| APAC     | Add PREG to ACC   | 1      | 1     |
| LPH      | Load high PREG, direct or indirect  | 1      | 1     |
| LT       | Load TREG, direct or indirect   | 1      | 1     |
| LTA      | Load TREG and accumulate previous product, direct or indirect             | 1      | 1     |
| LTD      | Load TREG, accumulate previous product, and move data, direct or indirect | 1      | 1     |
| LTP      | Load TREG and store PREG in accumulator, direct or indirect               | 1      | 1     |
| LTS      | Load TREG and subtract previous product, direct or indirect               | 1      | 1     |
| MAC      | Multiply and accumulate, direct or indirect                               | 2      | 3     |
| MACD     | Multiply and accumulate with data move, direct or indirect                | 2      | 3     |
| MPY      | Multiply TREG by data value, direct or indirect                           | 1      | 1     |
|          | Multiply TREG by 13-bit constant, short immediate                         | 1      | 1     |
| MPYA     | Multiply and accumulate previous product, direct or indirect              | 1      | 1     |
| MPYS     | Multiply and subtract previous product, direct or indirect                | 1      | 1     |
| MPYU     | Multiply unsigned, direct or indirect                                     | 1      | 1     |
| PAC      | Load ACC with PREG  | 1      | 1     |
| SPAC     | Subtract PREG from ACC  | 1      | 1     |
| SPH      | Store high PREG, direct or indirect                                       | 1      | 1     |
| SPL      | Store low PREG, direct or indirect  | 1      | 1     |
| SPM      | Set product shift mode  | 1      | 1     |
| SQRA     | Square and accumulate previous product, direct or indirect                | 1      | 1     |
| SQRS     | Square and subtract previous product, direct or indirect                  | 1      | 1     |

**Tabela 11.4: Ukazi za množenje in manipuliranje s TREG in PREG**

| MNEMONIK | OPIS   | BESEDE | CIKLI |
|----------|--|--------|-------|
| B        | Branch unconditionally, indirect             | 2      | 4     |
| BACC     | Branch to address specified by ACC           | 1      | 4     |
| BANZ     | Branch on current AR not 0, indirect         | 2      | 4     |
| BCND     | Branch conditionally                         | 2      | 4     |
| CALA     | Call subroutine at location specified by ACC | 1      | 4     |
| CALL     | Call subroutine, indirect                    | 2      | 4     |
| CC       | Call conditionally                           | 2      | 4     |
| INTR     | Soft interrupt                               | 1      | 4     |
| NMI      | Nonmaskable interrupt                        | 1      | 4     |
| RET      | Return from subroutine                       | 1      | 4     |
| RETC     | Return conditionally                         | 1      | 4     |
| TRAP     | Software interrupt                           | 1      | 4     |

**Tabela 11.5: Razvejvitveni ukazi**

| MNEMONIK | OPIS  | BESEDE | CIKLI |
|----------|---|--------|-------|
| BLDD     | Block move from data memory to data memory, direct/indirect with long immediate source      | 2      | 3     |
|          | Block move from data memory to data memory, direct/indirect with long immediate destination | 2      | 3     |
| BLPD     | Block move from program memory to data memory, direct/indirect with long immediate source   | 2      | 3     |
| DMOV     | Data move in data memory, direct or indirect  | 1      | 1     |
| IN       | Input data from I/O location, direct or indirect  | 2      | 2     |
| OUT      | Output data to port, direct or indirect   | 2      | 3     |
| SPLK     | Store long immediate to data memory location, direct or indirect                            | 2      | 2     |
| TBLR     | Table read, direct or indirect  | 1      | 3     |
| TBLW     | Table write, direct or indirect   | 1      | 3     |

**Tabela 11.6: Ukazi za manipulacijo s pomnilnikom ter vhodi in izhodi**

| MNEMONIK | OPIS  | BESEDE | CIKLI |
|----------|---|--------|-------|
| BIT      | Test bit, direct or indirect                        | 1      | 1     |
| BITT     | Test bit specified by TREG, direct or indirect      | 1      | 1     |
| CLRC     | Clear C bit   | 1      | 1     |
|          | Clear CNF bit                                       | 1      | 1     |
|          | Clear INTM bit                                      | 1      | 1     |
|          | Clear OVM bit                                       | 1      | 1     |
|          | Clear SXM bit                                       | 1      | 1     |
|          | Clear TC bit  | 1      | 1     |
|          | Clear XF bit  | 1      | 1     |
| IDLE     | Idle until interrupt                                | 1      | 1     |
| LDP      | Load data page pointer, direct or indirect          | 1      | 2     |
|          | Load data page pointer, short immediate             | 1      | 2     |
| LST      | Load status register ST0, direct or indirect        | 1      | 2     |
|          | Load status register ST1, direct or indirect        | 1      | 2     |
| NOP      | No operation  | 1      | 1     |
| POP      | Pop top of stack to low ACC                         | 1      | 1     |
| POPD     | Pop top of stack to data memory, direct or indirect | 1      | 1     |
| PSHD     | Push data memory value on stack, direct or indirect | 1      | 1     |
| PUSH     | Push low ACC onto stack                             | 1      | 1     |
| RPT      | Repeat next instruction, direct or indirect         | 1      | 1     |
|          | Repeat next instruction, short immediate            | 1      | 1     |
| SETC     | Set C bit   | 1      | 1     |
|          | Set CNF bit   | 1      | 1     |
|          | Set INTM bit  | 1      | 1     |
|          | Set OVM bit   | 1      | 1     |
|          | Set SXM bit   | 1      | 1     |
|          | Set TC bit  | 1      | 1     |
|          | Set XF bit  | 1      | 1     |
| SPM      | Set product shift mode                              | 1      | 1     |
| SST      | Store status register ST0, direct or indirect       | 1      | 1     |
|          | Store status register ST1, direct or indirect       | 1      | 1     |

**Tabela 11.7: Krmilni ukazi**

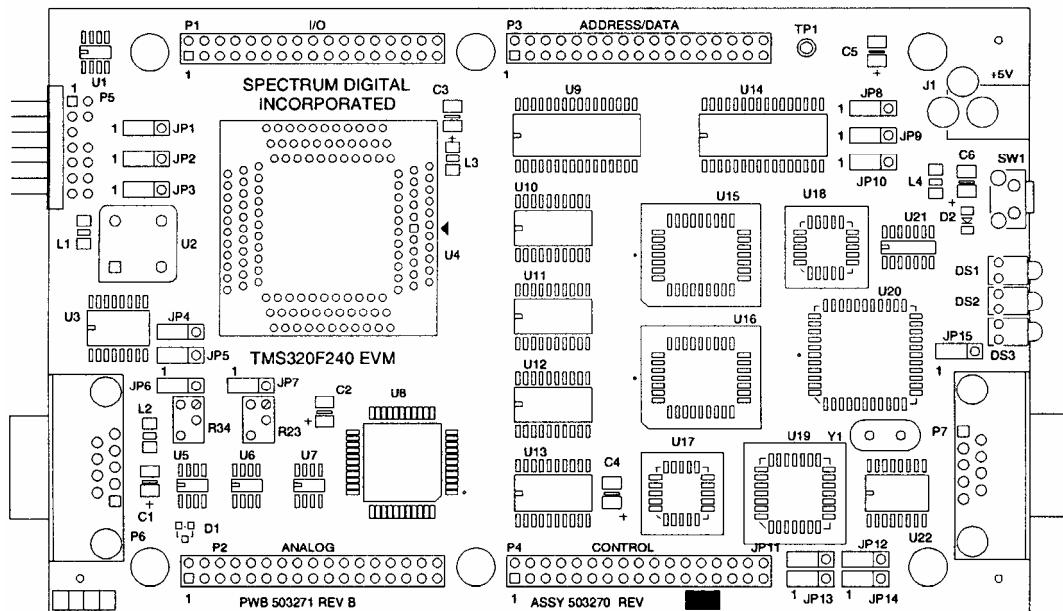
### 11.3.1.5 Razvojna orodja

Texas Instruments in nekateri drugi proizvajalci (angl. third parties) ponujajo celo vrsto razvojnih orodij za DSP TMS320F240. Tukaj bomo predstavili le razvojni modul (EVM) firme Spectrum Digital Inc. [32]. Slike 11.11 in 11.12 kažeta izgled in blokovno shemo modula.

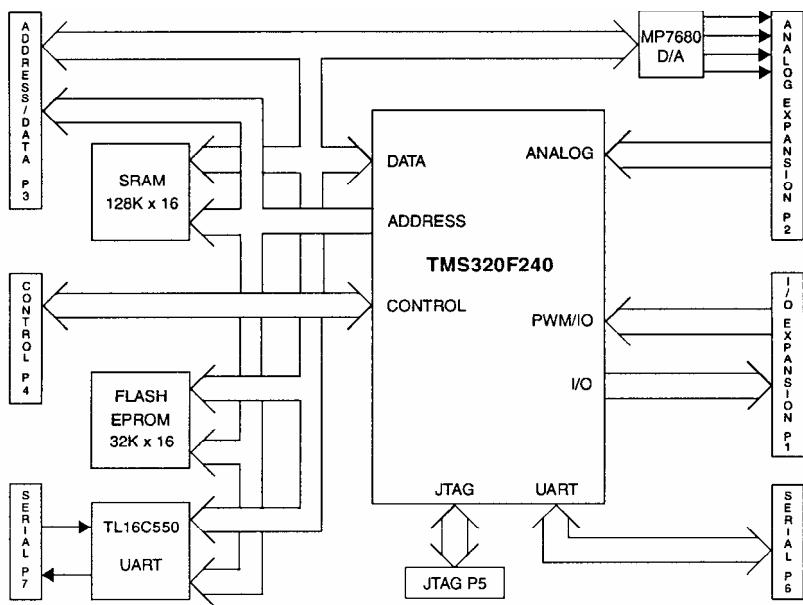
Na ploščici se poleg DSP-ja nahajajo naslednje enote:

- 128 Kword statični RAM pomnilnik (brez čakalnih stanj),
- 4-kanalni 12-bitni D/A pretvornik (MP7680),
- UART z gonilniki za EIA232,
- 32 Kword Flash EPROM,
- štirje razširitveni konektorji (podatkovno, naslovno, krmilno in I/O vodilo),
- IEEE 1149.1 JTAG konektor.

Programiranje krmilnika poteka s pomočjo prevajalnika za jezik C ter zbirnika v “off-line” režimu. Delo v “on-line” režimu pa omogoča nadzornik/razhroščevalnik “Source Debugger” v Windows okolju. Program dovoljuje hkratno spremljanje več podatkovnih in programskeih pomnilniških blokov ter registrov.



Slika 11.11: EVM za TMS320F240



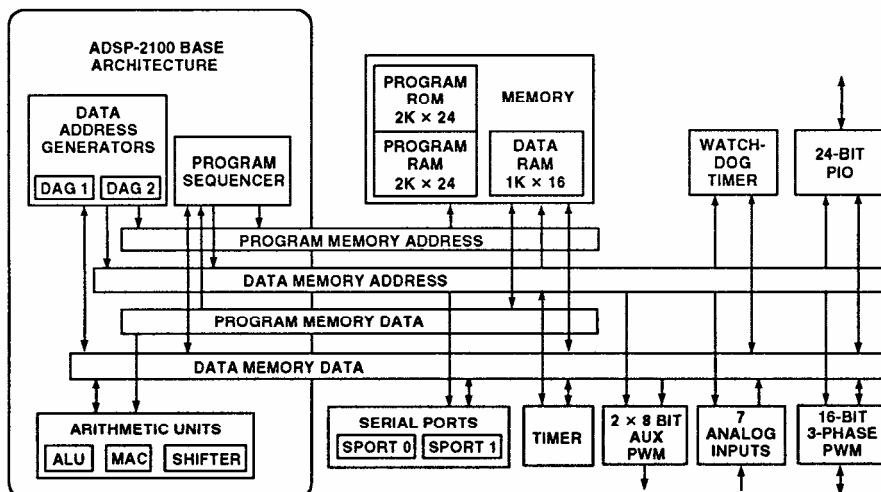
Slika 11.12: Blokovna shema EVM za TMS320F240

### 11.3.2 DSP mikrokrmilnik ADMC331

ADMC331 (Analog Devices) je mikrokrmilnik, ki je, podobno kot TMS320F240, namenjen krmiljenju in regulaciji pogonov z asinhronskimi, sinhronskimi, elektronsko komutiranimi in reluktančnimi motorji v aplikacijah kot so pralni stroji, črpalke, kompresorji v hladilnikih, ventilatorske naprave, industrijski pogoni s spremenljivo hitrostjo itd. Krmilnik (slika 11.13)

je zasnovan na DSP jedru ADSP-2100 (minimalni čas izvajanja vsakega ukaza ob neposrednem naslavljanju je 35 ns), ki vsebuje neodvisne enote za aritmetično-logične operacije (ALU), množenje in akumuliranje (MAC) in pomik (SHIFTER). Programsko (RAM, 2 K x 24 bit in ROM, 2 K x 24 bit) in podatkovno (RAM, 1 K x 16 bit) pomnilniško področje sta ločeni in povezani z ostalimi podsestavi prek lastnih podatkovnih<sup>2</sup> in naslovnih vodil.

Ostali funkcionalni deli ADMC331 so podobni kot pri konkurenčnih mikrokrmlnikih enake namembnosti: 7 A/D pretvornikov (ločljivost 12 bitov), enota za generiranje PWM, enote za asinhronski (UART) in sinhronski serijski prenos, 24 binarnih vhodov/izhodov ter 16-bitna časovna straža (watchdog).



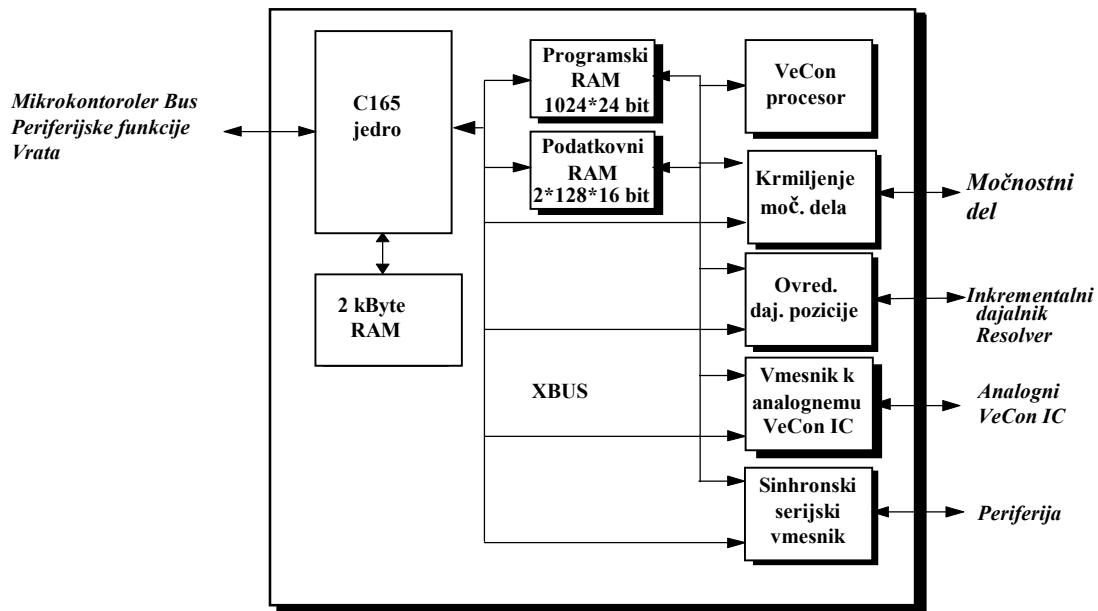
*Slika 11.13: Blokovna shema DSP mikrokrmlnika ADMC331*

### 11.3.3 Ve-Con

Omenimo tukaj še enega izmed prvih mikrokrmlniških sistemov za uporabo v močnostni elektroniki, ki le deloma spada med DSP krmilnike. VeCon (Vector Control) je nastal v prvi polovici devetdesetih kot rezultat sodelovanja štiridesetih velikih evropskih podjetij (ABB, AEG, Bosch, Grundig, Siemens...) [2]. Čeprav je bila v začetku ideja zasnovana na integraciji vseh funkcij na enem čipu, se je na koncu za tehnološko in finančno učinkovitejšo izkazala izvedba na dveh čipih: digitalnem in analognem.

Digitalni VeCon (slika 11.14) je v osnovi dvoprocesorski čip z dodatnim pomnilnikom in vhodno-izhodnimi enotami. 16-bitni Siemensov klasični mikrokrmlnik C165 skrbi za zunano regulacijsko zanko (regulacija pozicije, hitrosti in fluksa) ter krmiljenje, posluževanje in komunikacijo z drugimi sistemi. Programiranje je izvedljivo tudi v višjih jezikih.

<sup>2</sup> Dolžina ukazne (programske) besede je 24 bitov, podatkovne pa 16 bitov.



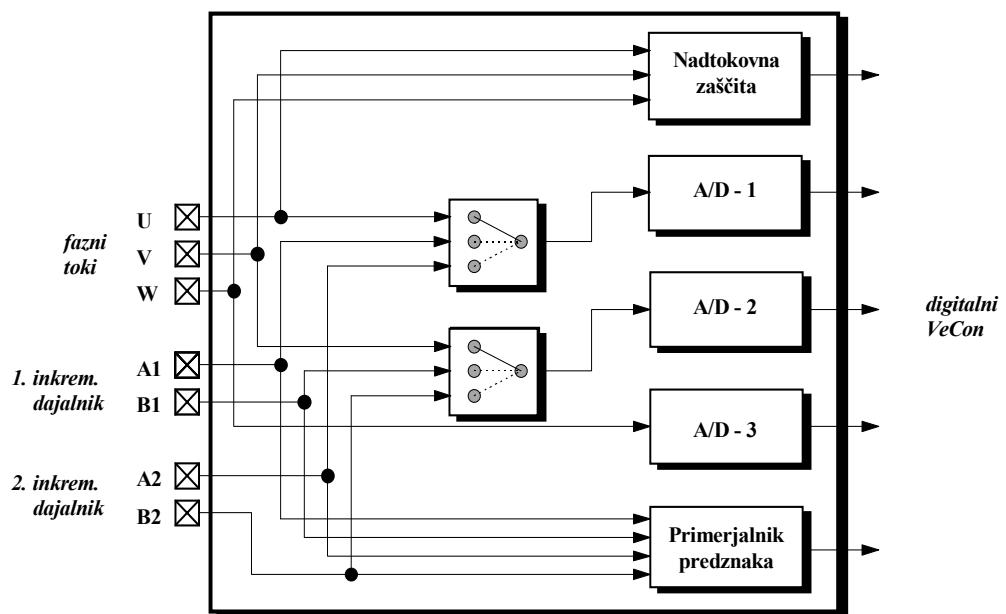
*Slika 11.14: Digitalno integrirano vezje VeCon*

Regulacijski program izvaja na DSP baziran 16-bitni regulacijski procesor VeCon. Poleg standardnih operacij vsebuje tudi tabelo za hitro računanje trigonometričnih funkcij (transformacije med koordinatnimi sistemi in ovrednotenje dajalnika pozicije).

Oba procesorja imata dostop do skupnega 24-bitnega programskega in 16-bitnega podatkovnega Dual Port RAM-a preko XBUS-a.

Periferne enote krmilijo močnostni del pogona (preko PWM), zajemajo podatke iz dajalnika pozicije, izmenjujejo podatke z analognim čipom VeCon in komunicirajo z dodatno periferijo (npr. serijska komunikacija z A/D in D/A pretvorniki).

Poseben čip, analogni VeCon (slika 11.15), je namenjen obdelavi analognih signalov iz treh merilnikov faznih tokov ter iz dveh inkrementalnih dajalnikov. Slednja lahko generirata signale bodisi v analogni ali v digitalni obliki. Čip tudi zaznava, ali je vrednost toka presegla dovoljeno mejo.



Slika 11.15: Analogno VeCon integririrano vezje

## 12. PERIFERNA INTEGRIRANA VEZJA

Videli smo, da vsebujejo mikroracunalniki poleg CPU-ja oz. mikroprocesorja še RAM pomnilnik (včasih tudi ROM oz. EPROM), binarne vhodno/izhodne enote, komunikacijsko enoto, nekateri celo A/D pretvornike ali pomožne procesne enote (npr. TPU pri MC68332 ali matematični koprocesor). Pri reševanju določenega problema moramo pogosto vključiti dodatna digitalna in analogna vezja. V tem poglavju si bomo skozi praktične primere ogledali nekaj integriranih vezij, ki jih uporabljamo za regulacijo procesov v realnem času.

### 12.1 Pomnilniki

O pomnilnikih smo že govorili v 2. poglavju. Tukaj bomo pokazali nekaj standardnih čipov ter na kratko opisali t.i. flash pomnilnike.

#### 12.1.1 EPROM

Na sliki 12-1 je prikazana blokovna shema 128 Kbytnega EPROM pomnilnika. Na čipu se poleg podatkovnih nožic (DQ7 - DQ0) nahaja še 17 naslovnih nožic (A16-A0). Čip napajamo z napetostjo  $V_{CC} = 5$  V, za programiranje je zadolžena napetost na nožici  $V_{PP}$ , masa pa je označena z  $V_{SS}$  (glej tudi sliko 12.2) Poleg tega so na čipu tudi trije krmilni signali<sup>1</sup>:

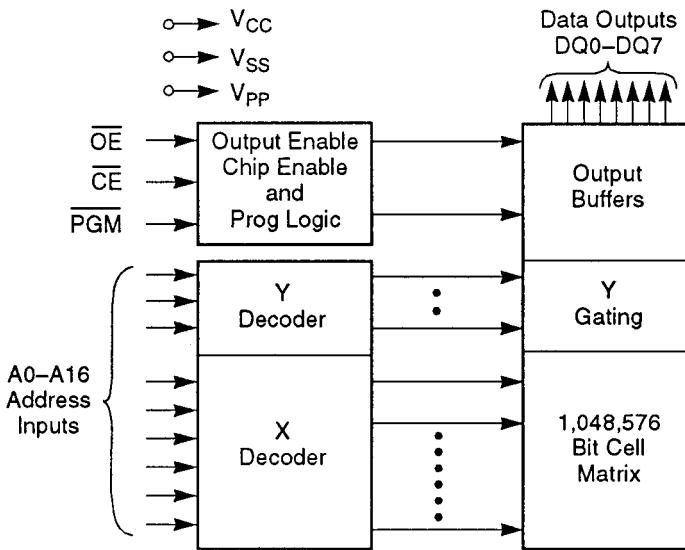
- CE (E) - izbira čipa (angl. chip enable) poznamo že kot CS vhod.
- OE (G) - sprostitev izhodov (angl. output enable) omogoča čitanje podatkov iz EPROMA. Povežemo ga z nožico R/W računalnika. Ob neaktivnem OE je podatkovno vodilo v stanju visoke impedance.
- PGM (P) - omogočanje programiranja (angl. program enable).

Opis osnovnih operacij:

1. Brisanje vsebine EPROM-a dosežemo z ultravijoličasto svetlobo (valovna dolžina 253,7 nm, osvetljenost  $12000 \mu\text{W}/\text{cm}^2$  v trajanju 15-20 minut - velja za Am27x010). Po brisanju se vsi biti v EPROM-u postavijo na "1".
2. Za programiranje (vpisovanje) nove vsebine je potrebno pripeljati 12,75 V na nožico  $V_{PP}$ , aktivirati CE in PGM ter deaktivirati OE. Novi podatki prihajajo po podatkovnem vodilu. Po programiranju je možno naloženo vsebino preveriti, kar nam omogoča poseben EPROM programator.
3. Normalno obratovanje: čip je nameščen v ciljnem računalniku in deluje kot običajno brialno pomnilniško vezje.

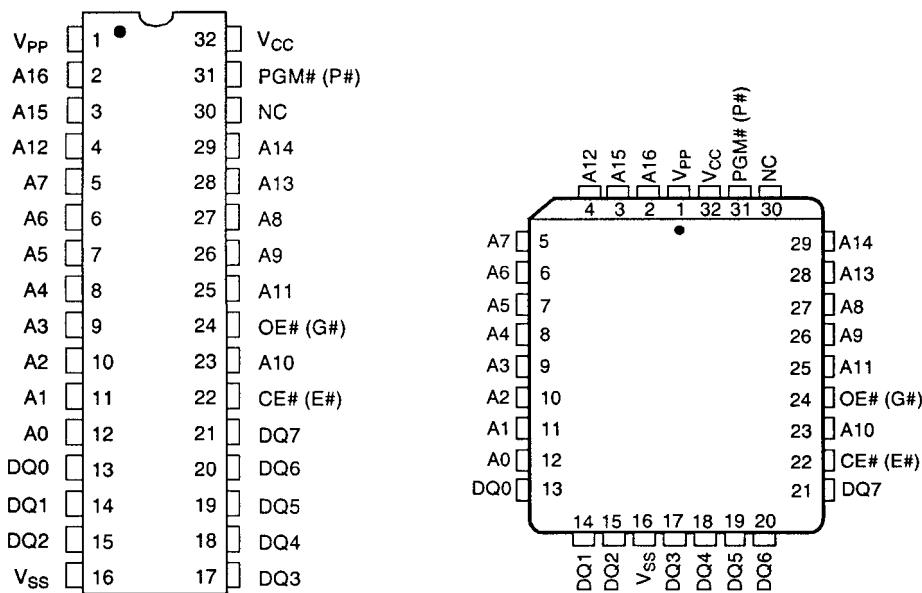
---

<sup>1</sup> V oklepajih so zapisane oznake po JEDEC nomenklaturi.



**Slika 12.1: Blokovna shema EPROM-a Am27x010 (1 Mbit - 128 K x 8 bit) s dostopnim časom 50 ns**

Slika 12.2 kaže oblike ohišij EPROM-a Am27x010. **DIP** (Dual In-Line Package) je trenutno najpogosteša izvedba ohišij. Nožice (maksimalno število 48) so postavljene navpično na površino čipa. Razmik med nožicami je 0,1 inča (2,54 mm). **PLCC** (Plastic Leaded Chip Carrier) ohišje je bistveno primernejše za čipe z veliko nožicami, saj so le-te rasporejene ob vseh stranicah čipa. Nožice so zavihane navznoter, čip se pa namesti na posebno podnožje.



**Slika 12.2: DIP in PLCC ohišja z oznakami nožic**

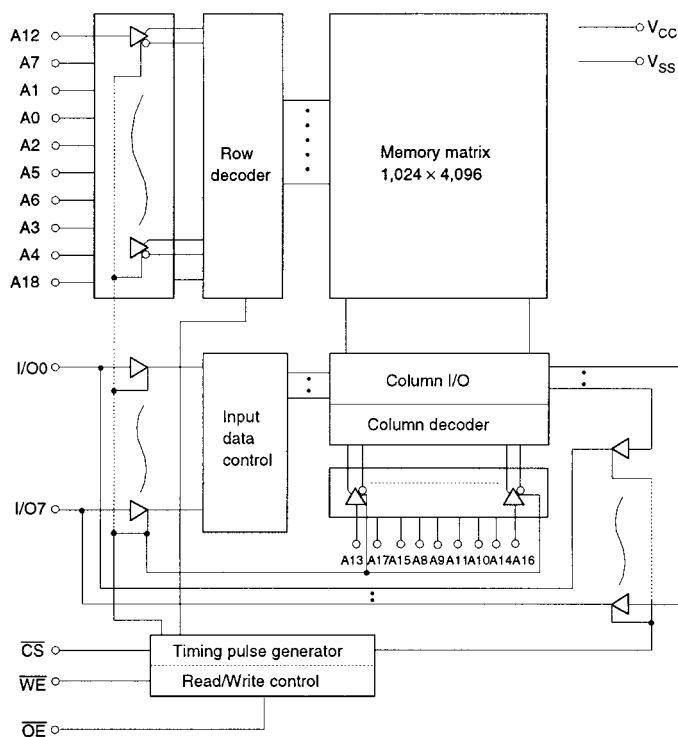
### 12.1.2 RAM

Slika 12.3 kaže primer RAM-a z veliko kapaciteto. Večino nožic smo že spoznali pri EPROM pomnilniku. Razlike:

- nožice podatkovnega vodila so označene z I/O (input/output), saj je pretok podatkov dvosmeren,
- WE omogoča branje RAM-a (povežemo ga z R/W izhodom mikroračunalnika).

Z različnimi kombinacijami krmilnih nožic izberemo načine obratovanja:

| <b>WE</b> | <b>CS</b> | <b>OE</b> | <b>način</b>    | <b>podatkovno vodilo</b> |
|-----------|-----------|-----------|-----------------|--------------------------|
| x         | 1         | x         | ni izbran       | visoka impedanca         |
| 1         | 0         | 1         | izhod neaktivен | visoka impedanca         |
| 1         | 0         | 0         | branje          | izhodni podatki          |
| 0         | 0         | 1         | vpisovanje      | vhodni podatki           |
| 0         | 0         | 0         | vpisovanje      | vhodni podatki           |



*Slika 12.3: RAM pomnilnik Hitachi HM628512AI (512 Kbyte) s časom dostopa 70 ns*

### 12.1.3 Flash EPROM pomnilniki

Flash EPROM je poseben primer *permanentnih pomnilnikov* (angl. non-volatile memory). V bistvu gre za EEPROM, ki ga enostavno brišemo in reprogramiramo kar s samim mikroračunalnikom.

Flash pomnilniki (npr. Texas Instruments TMS28F400) se vse pogosteje pojavljajo v računalniških, komunikacijskih in industrijskih aplikacijah. Ponavadi vsebujejo BIOS (angl. Basic Input/Output System: najosnovnejši vmesnik k periferiji in nadzor samozagona - angl. bootstrap), pri perifernih enotah, kot so npr. tiskalniki, pa so v flash-u shranjeni tipografija (fonti) in formati. Ta tip pomnilnika se uveljavlja zlasti pri prenosnih elektronskih napravah, kot so mobilni telefoni, digitalne kamere, kjer jih uporabljamo za shranjevanje podatkov, pošiljanje in sprejemanje faksov ter za shranjevanje digitalnih avdio in video zapisov. Značilnosti teh aplikacij so zahteve po čim večji gostoti informacij na pomnilnik. Zaradi baterijskega napajanja so nekateri tipi pomnilnika grajeni za specifične napajalne napetosti in majhno porabo.

Na trgu trenutno najdemo flash pomnilnike kapacitete 2 Mbit, 4 Mbit in 8 Mbit z dostopnimi časi od 100 ns do 60 ns, kar je le nekoliko več od dostopnega časa RAM pomnilnikov. Tipične napajalne napetosti ( $V_{PP}$  in  $V_{CC}$ ) so 3,3 V (za baterijsko napajane naprave), 5 V in 12 V.

Tukaj bomo na kratko opisali flash s t.i. "boot block arhitekture". Pomnilnik lahko obratuje v treh režimih: ERASE (brisanje<sup>2</sup>), WRITE (vpisovanje) ali READ (branje).

V flash-u ni mogoče brisati posameznih pomnilniških lokacij. Boot block arhitektura omogoča brisanje samo posameznih različno velikih blokov pomnilnika (glej tudi sliko 12.4). Možnost ločenega brisanja podaljšuje življenjsko dobo pomnilnika. Pri operacijah pisanja in branja lahko naslavljamo posamezne lokacije. Poseben "boot" blok pomnilnika je namenjen za shranjevanje programa, ki je potreben za osnovno začetno delovanje naprave z vgrajenim pomnilnikom. Zaradi tega je njegovo nemerno brisanje onemogočeno s hardversko zaščito in ga lahko opravimo le s povišano napetostjo. "Boot" blok je običajno na začetku ali na koncu naslovljivega področja flash-a, npr. za RISC procesorje [27 ].

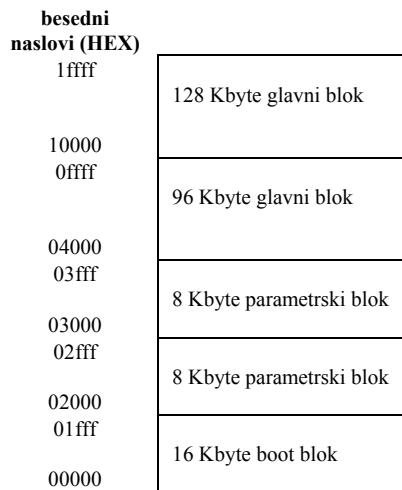
Tukaj si bomo ogledali flash EPROM MT28F400B3xx-xxB firme Micron. Nekatere osnovne značilnosti:

- Kbyte/8 Kword boot blok,
- dva 8 Kbyte/4 Kword parametrskih blokov,
- dva bloka glavnega pomnilnika,
- čas dostopa 60 ns/80 ns,
- možnost 100 000 brisanj,
- možnost organizacije bytnega ali besednega formata.

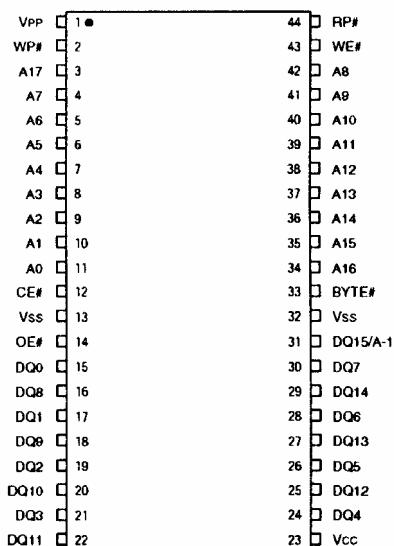
Slike 12.4 in 12.5 kažeta pomnilniško mapo in razporeditev nožic v SOP izvedbi.

---

<sup>2</sup> Brisanje pomeni postavljenje dela ali celotnega pomnilnika v vnaprej določeno stanje.



Slika 12.4: Pomnilniška mapa MT28F400B3xx-xxB



Slika 12.5: SOP izvedba čipa

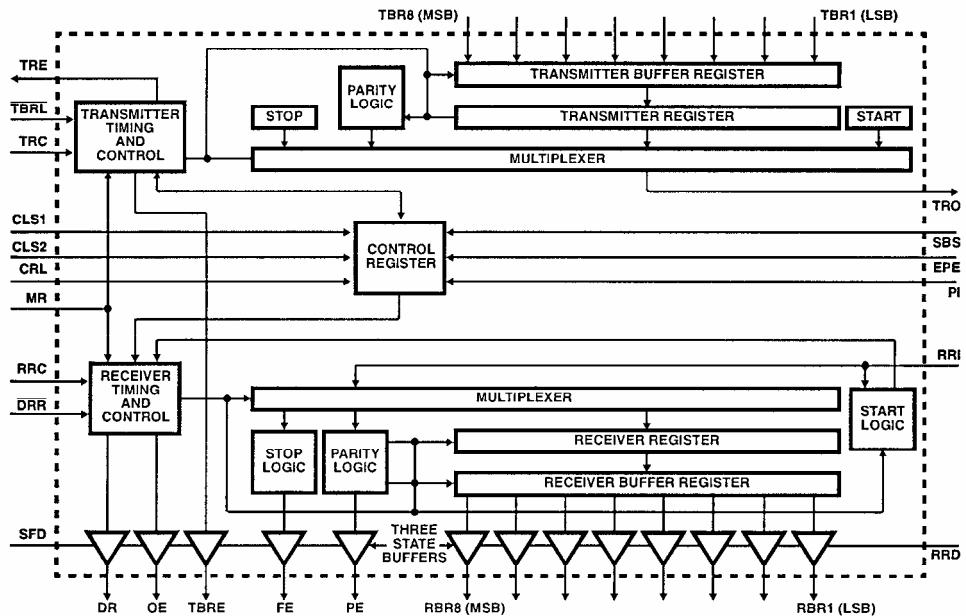
Poleg že znanih nožic vsebuje ta pomnilnik še vhoda RP (Reset/Power Down), WP (Write Protect) ter VPP (napajalna napetost za pisanje in brisanje). Posamezne operacije izbiramo z ustrezeno nastavitevjo krmilnih in napajalnih nožic ter s setiranjem ustreznih bitov na podatkovnem vodilu.

Veliko proizvajalcev mikroprocesorskih komponent ponuja mikrokrmilnike, ki na čipu poleg perifernih enot vsebujejo tudi flash pomnilnik.

## 12.2 Komunikacijski vmesniki in gonilniki

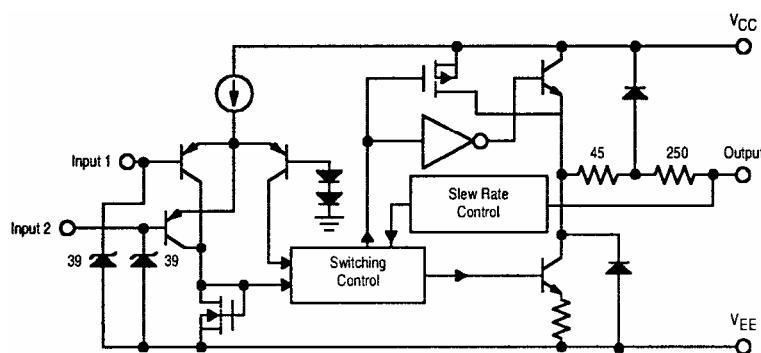
O funkciji asinhronskega zaporednega vmesnika (UART), ki je del mikrokrmilnikov, smo govorili že v prejšnjih poglavjih. Na sliki 12.6 je prikazan primer UART-a Intersil CDP6402 s hitrostjo prenosa do 200 Kbps pri 5 V. Izhodne podatke iz mikroračunalnika pripeljemo prek

8-bitnega vodila na vhode TBR8 - TBR1 (serijski izhod TRO), vhodne serijske podatke v mikroračunalnik pa dobimo na nožicah podatkovnega vodila RBR8 - RBR1 (serijski vhod RRI). Parametre prenosa (število podatkovnih bitov, tip in število paritetnih bitov ter število STOP bitov) določamo s postavitevijo bitov CLS2, CLS1, PI, EPE in SBS v predhodno določena stanja. Parity error bit (PE) v stanju 1 označuje napako pri prenosu.



**Slika 12.6: UART CDP6402**

Izhodni signali iz mikroprocesorja imajo TTL nivoje (0 V - 5 V, pozitivna logika). Po drugi strani smo pri EIA-232 standardu govorili o drugačnih napetostnih nivojih (v tem primeru  $\pm 5$  V do  $\pm 15$  V, negativna logika). Za usklajevanje teh dveh različnih tipov napetostnih območij potrebujemo posebna vezja, kot je linijski gonilnik Motorola MC14C88B (slika 12.7). V tem primeru lahko pripeljemo štiri signale TTL nivoja (INPUT), ki jih vezje prilagodi nivojem standarda EIA-232 (OUTPUT). Na MC14C88B lahko pripeljemo tudi signale CMOS nivoja, temu ustrezno pa moramo prilagoditi napajanja na nožicah  $V_{EE}$  in  $V_{CC}$ .



**Slika 12.7: Linijski gonilnik MC14C88B**

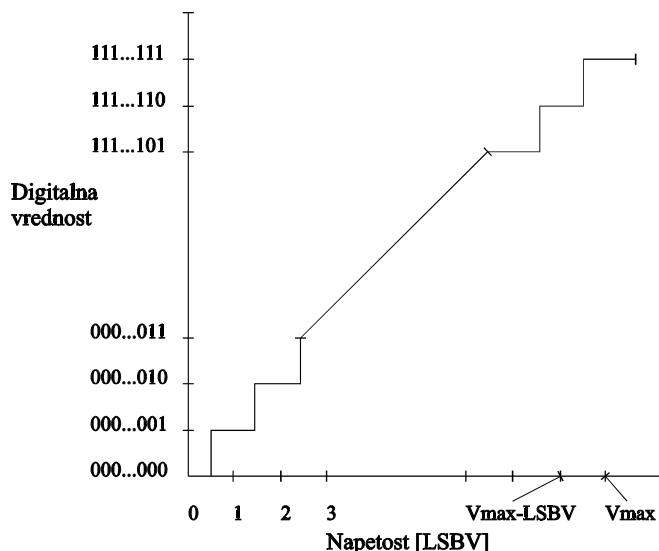
V primeru zajemanja signalov v obratni smeri potrebujemo gonilnik MC14C89B ali podobno vezje.

### 12.3 Digitalno/analogni in analogno/digitalni pretvorniki

Digitalno/analogni (D/A) in analogno/digitalni (A/D) pretvorniki so ene najbolj pogostih vhodno/izhodnih enot procesnih mikroračunalnikov. Njihova naloga je prilaganje digitalne informacije v mikroračunalniku zunanjim analognim napetostim ali tokom ali ibratno. Pri izbiri ustreznega pretvornika izhajamo iz naslednjih kriterijev:

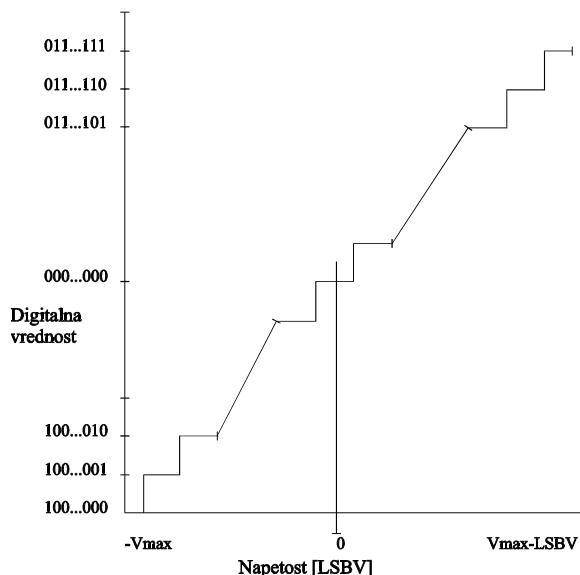
- **Ločljivost (digitalno območje)**: pri sistemih, ki ne zahtevajo velike natančnosti prikaza analogne vrednosti, običajno izberemo 8-bitne pretvornike (ločljivost 1/256), pri zahtevnejših pa vsaj 10- (ločljivost 1/1024) ali 12-bitne pretvornike (ločljivost 1/4096). Večje ločljivosti pri hitrih regulacijskih sistemih navadno ne pridejo v poštev zaradi neskladja z ostalimi kriteriji.
- **Hitrost pretvorbe**: ta je pri D/A pretvornikih nekaj  $\mu$ s, pri A/D pretvornikih pa nekoliko večja.
- **Število kanalov na čipu**: proizvajalci ponujajo čipe z enim, dvema ali štirimi analognimi kanali (izjemoma tudi več).
- **Cena** komponent narašča v odvisnosti od zgoraj opisanih lastnosti in je za kvalitetne pretvornike lahko celo višja od cen mikroprocesorjev. Zato se odločamo za kompromis med ceno in lastnostmi.
- **Analogno območje** je odvisno od zahtev procesa: pri analognih vhodih od izhodne napetosti merilnikov, pri izhodih od ojačevalnikov. Običajna območja so  $\pm 10$  V,  $\pm 5$  V, (*bipolarni pretvorniki*) oz. 0 V - 2,5 V, 0 V - 4,096 V, 0 V - 5 V (t.i. *unipolarni pretvorniki*). Večina novejših pretvornikov omogoča spremembo notranje reference ali priključitev zunanje referenčne napetosti. S tem dosežemo fleksibilno določanje limitnih vrednosti.
- **Tip prenosa digitalnih podatkov** znotraj mikroračunalnika: **serijski prenos** zahteva manj povezav, zato pa so hitrosti prenosa manjše kot pri pretvornikih s **paralelnim prenosom**, **ti pa imajo** bistveno večje število povezav. Serijski prenos postaja vse zanimivejši zaradi povečanja hitrosti prenosa in enostavnosti vgradnje v že obstoječi mikroračunalnik (glej pogl. 6.2.1).

Unipolarnost ali bipolarnost (sliki 12.8 in 12.9) je zelo pomembna pri tolmačenju digitalnega ekvivalenta analogne vrednosti ter njegovem prilaganju logiki dvojiškega komplementa. Na slikah je  $V_{max}$  maksimalna vrednost,  $LSBV$  pa ločljivost napetosti oz. napetost, ki ustrezata enemu bitu. Za prvi primer velja  $LSBV = 5 \text{ V}/4096 = 1,22 \text{ mV}$  pri  $V_{ref} = 5 \text{ V}$  in 12-bitnem pretvorniku.



**Slika 12.8: Unipolarna karakteristika**

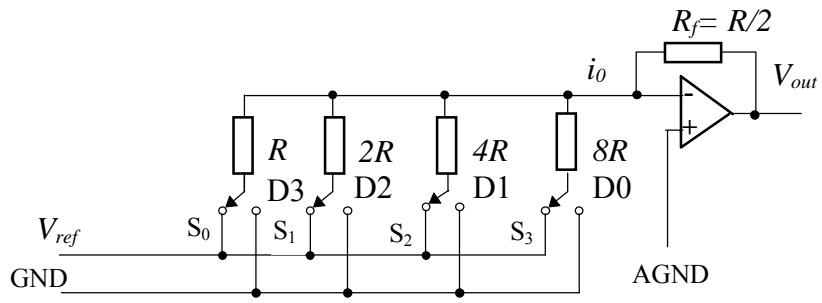
V drugem primeru sta z  $+V_{max}$  in  $-V_{max}$  označeni maksimalni vrednosti. Pri  $V_{ref} = 5$  V in 12-bitnem pretvorniku velja  $LSBV = 2 \cdot 5\text{ V} / 4096 = 2,44$  mV. Ponovno lahko opazimo nesimetrijo med pozitivno in negativno napetostjo, ki jo poznamo že iz obravnave dvojiškega komplementa.



**Slika 12.9: Bipolarna karakteristika**

### 12.3.1 Digitalno/analogni pretvorniki

Princip delovanja D/A pretvornika bomo zaradi enostavnosti prikazali na primeru 4-bitnega pretvornika (slika 12.10).



**Slika 12.10: Principialna zgradba 4-bitnega D/A pretvornika**

Pretvornik je sestavljen iz paralelnih vezanih uporov z različnimi težami (vsak naslednji je dvakrat večji od predhodnega) in operacijskega ojačevalnika. Podatkovni biti na digitalni strani (D3-D0) prožijo stikala, ki vežejo ustrezne upore na neko referenčno napetost ali maso (odvisno od stanja 1 ali 0). Skupni tok pretvornika je

$$\begin{aligned} i_0 &= \frac{V_{ref}}{R} D3 + \frac{V_{ref}}{2R} D2 + \frac{V_{ref}}{4R} D1 + \frac{V_{ref}}{8R} D0 = \\ &= \frac{2V_{ref}}{R} \left( \frac{D3}{2^1} + \frac{D2}{2^2} + \frac{D1}{2^3} + \frac{D0}{2^4} \right) = \frac{2V_{ref}}{R} D, \end{aligned}$$

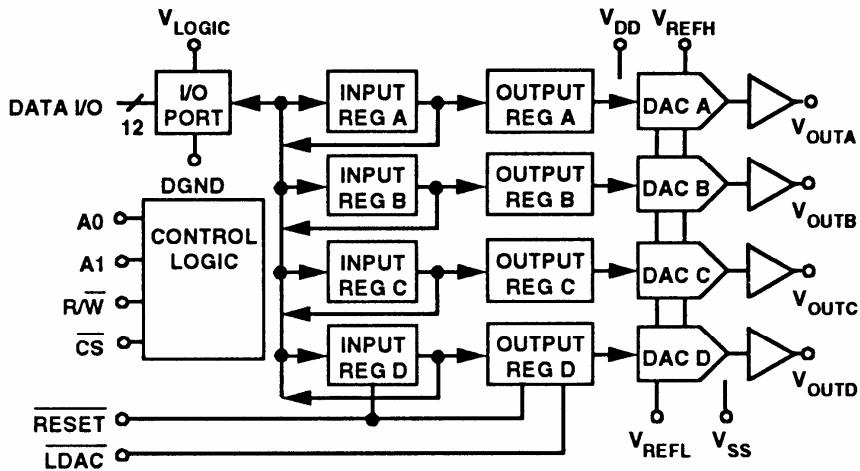
kjer je z  $D$  označen izraz

$$D = \frac{D3}{2^1} + \frac{D2}{2^2} + \frac{D1}{2^3} + \frac{D0}{2^4}, D_n = \begin{cases} 0 \\ 1 \end{cases}.$$

Izhodna napetost z ozirom na analogno maso (AGND) je

$$v_{out} = -i_0 R_f = -V_{ref} D.$$

Konkretna serijska D/A in A/D pretvornika smo si ogledali v poglavju 6.2.1, primer paralelnega D/A pretvornika pa kaže slika 12.11. DAC8412/DAC8413 firme Analog Devices je 12-bitni D/A pretvornik s štirimi izhodi. Spodnjo in zgornjo mejo referenčne napetosti določata nožici  $V_{REFL}$  in  $V_{REFH}$ . Za izbiro izhoda uporabljamo naslovna vhoda A0 in A1 ( $2^2 = 4$ ), izbiro čipa pa opravimo z vhodom CS. Poleg povezave s podatkovnim vodilom (DATA I/O) mora procesor poskrbeti za signal za pisanje/branje (R/W). Podatki iz vmesnih vhodnih registrov se na izhode prenesejo ob aktivnem signalu LDAC.



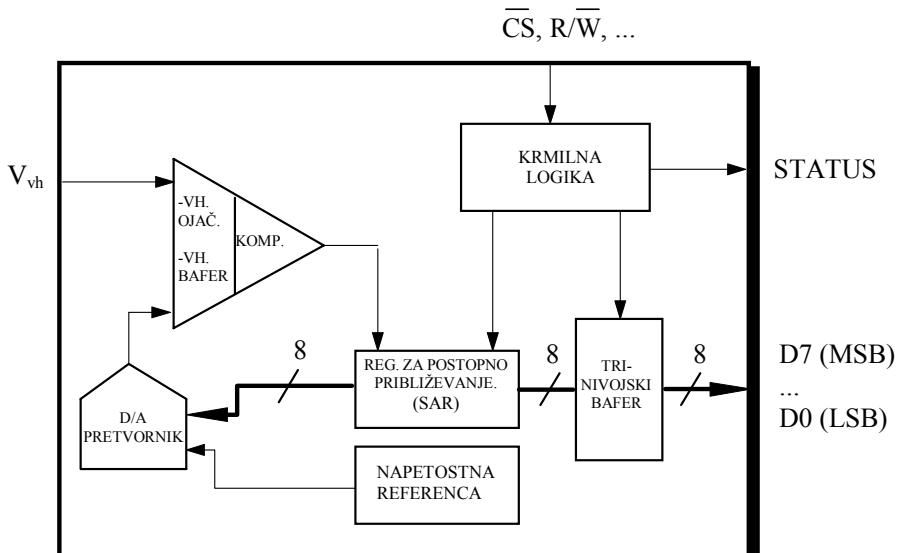
Slika 12.11: Blokovna shema D/A pretvornika DAC8412/8413 firme Analog Devices

Nekatere značilnosti DAC8412/8413:

- napetostno območje od [0, +5] do  $\pm 15V$ ,
- unipolarno ali bipolarno obratovanje,
- štirje izhodi (naslavljamo jih z A0 in A1),
- možnost branja stanja registrov,
- čas pretvorbe 6  $\mu s$ .

### 12.3.2 Digitalno/analogni pretvorniki

Tukaj si bomo ogledali le primer A/D pretvornika, ki uporablja metodo postopnega približevanja (slika 12.12).

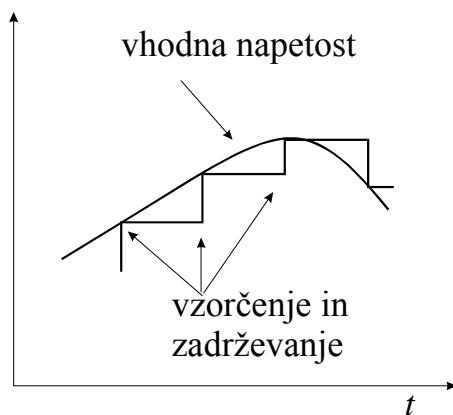


Slika 12.12: Blokovna shema 8-bitnega A/D pretvornika

Vhodno analogno napetost primerjamo z napetostjo iz internega D/A pretvornika. V odvisnosti od predznaka razlike se v posebnem registru (SAR - angl. Successive Approximation Register) postavi digitalna vrednost po metodi postopnega približevanja. Leto peljemo na D/A pretvornik, s čimer je zanka zaključena. Vsebina SAR-a se bo spremenjala, dokler obe napetosti na vhodu primerjalnika ne bosta enaki. Takrat je vrednost v SAR-u digitalni ekvivalent analogne napetosti.

Trajanje postopka postopnega približevanja je odvisno od ločljivosti pretvornika, načeloma se pri pretvornikih srednjega razreda giblje okrog nekaj  $\mu\text{s}$ .

Seveda se med postopkom iskanja digitalnega ekvivalenta vhodna napetost ne sme spremenjati (slika 12.13). Za to poskrbi vzorčevalno-zadrževalno vezje (S&H - angl. sample and hold).



**Slika 12.13: Vzorčenje in zadrževanje vhodne napetosti**

Slika 12.14 kaže primer 12-bitnega A/D pretvornika Analog Devices AD7874.

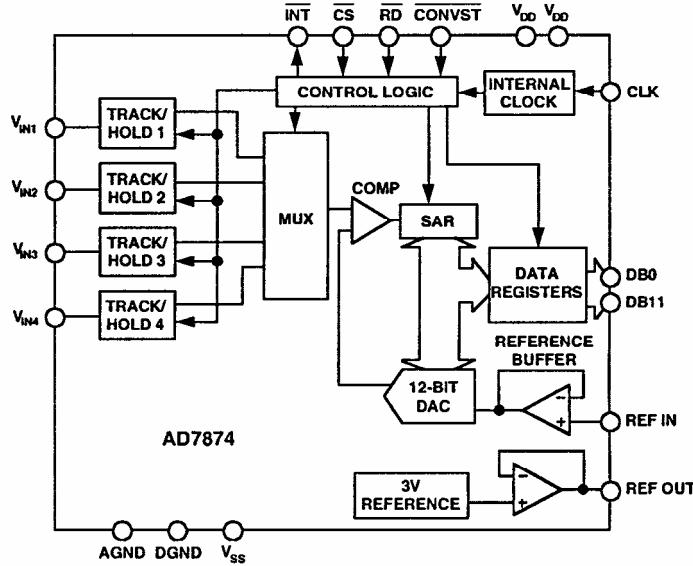
Njegove značilnosti so:

- vhodno napetostno območje  $\pm 10 \text{ V}$ ,
- hkratno vzorčenje in zadrževanje vrednosti vseh kanalov ( $2 \mu\text{s}$ ),
- čas pretvorbe je  $8 \mu\text{s}$  po kanalu.

Opis nekaterih nožic:

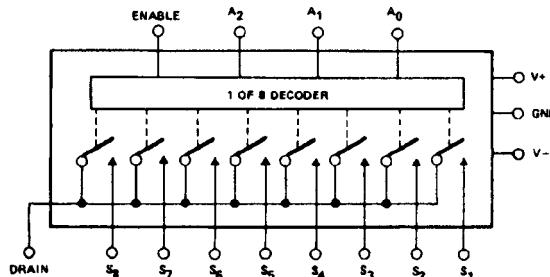
- CONVST (angl. conversion start) - vzorčenje in zadrževanje napetosti ter začetek pretvorbe,
- RD - branje digitalnega ekvivalenta,
- INT - inidikacija prvega branja.

Omenimo tukaj še posebnost AD7874. Čeprav ima pretvornik štiri vhodne kanale, na njem ne najdemo nožic za njihovo izbiro, kot so to na primer A0 in A1 pri DAC8412 (slika 12.11). Pri AD7874 se namreč ob vsakem branju (signal RD aktiven) naslovi naslednji vhodni kanal:  $V_{IN1}$ ,  $V_{IN2}$ ,  $V_{IN3}$ ,  $V_{IN4}$ ,  $V_{IN1}$ , ...



Slika 12.14: A/D pretvornik AD7874

V primerih, ko obstaja potreba po zajemanju več analognih signalov kot je na voljo A/D pretvornikov, si lahko pomagamo z analognimi multiplekserji, kot je npr. MUX-08 firme Analog Devices (slika 12.15).



Slika 12.15: Analogni multiplekser MUX-08

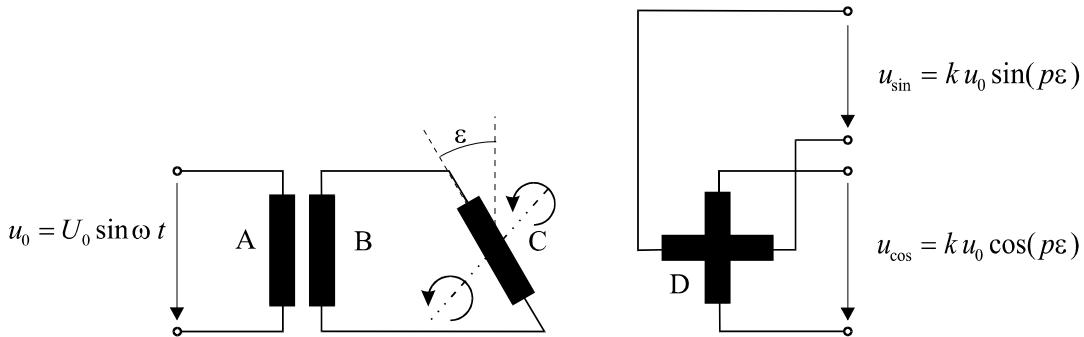
Multiplekser poveže enega izmed vhodnih kanalov S<sub>8</sub> - S<sub>1</sub> z nožico DRAIN ob aktivnem vhodu ENABLE. Za izbiro kanala so zadolžene nožice A<sub>2</sub> - A<sub>0</sub>.

## 12.4 Integrirana vezja za ugotavljanje kota zasuka in kotne hitrosti

V rotacijskih sistemih, ki jih krmili ali regulira mikroračunalnik, se največkrat pojavlja potreba po merjenju kota zasuka (pozicije) ali njegovega odvoda - kotne hitrosti. Večinoma gre za meritne elemente, ki so sestavljeni iz fiksnega in vrtečega se dela. Prvi je mehansko povezan s statorjem motorja, drugi pa je pritrjen na rotor motorja. Zaradi majhnih dimenzijs in mase (majhne vztrajnosti) taki meritniki ne vpivajo na mehanske lastnosti merjenca. Tukaj bomo prikazali dva tipa rotacijskih meritnikov ter posebna integrirana vezja za povezavo med meritnikom in mikroračunalnikom.

### 12.4.1 Resolverski digitalni pretvornik

Resolverje uporabljamo za natančno meritev kota rotorja. Eno od inačic kaže slika 12.16 [2]. V bistvu gre za miniaturni generator z enim navitjem na rotorju (C) ter dvema za  $90^\circ$  premaknjjenima sekundarnima navitjima na statorju (D)<sup>3</sup>.



**Slika 12.16: Principialna zgradba resolverja (p je število polovih parov resolverja)**

Primarno navitje napajamo z referenčno sinusno napetostjo visoke frekvence (npr. 20 kHz):

$$u_0 = U_0 \sin \omega t.$$

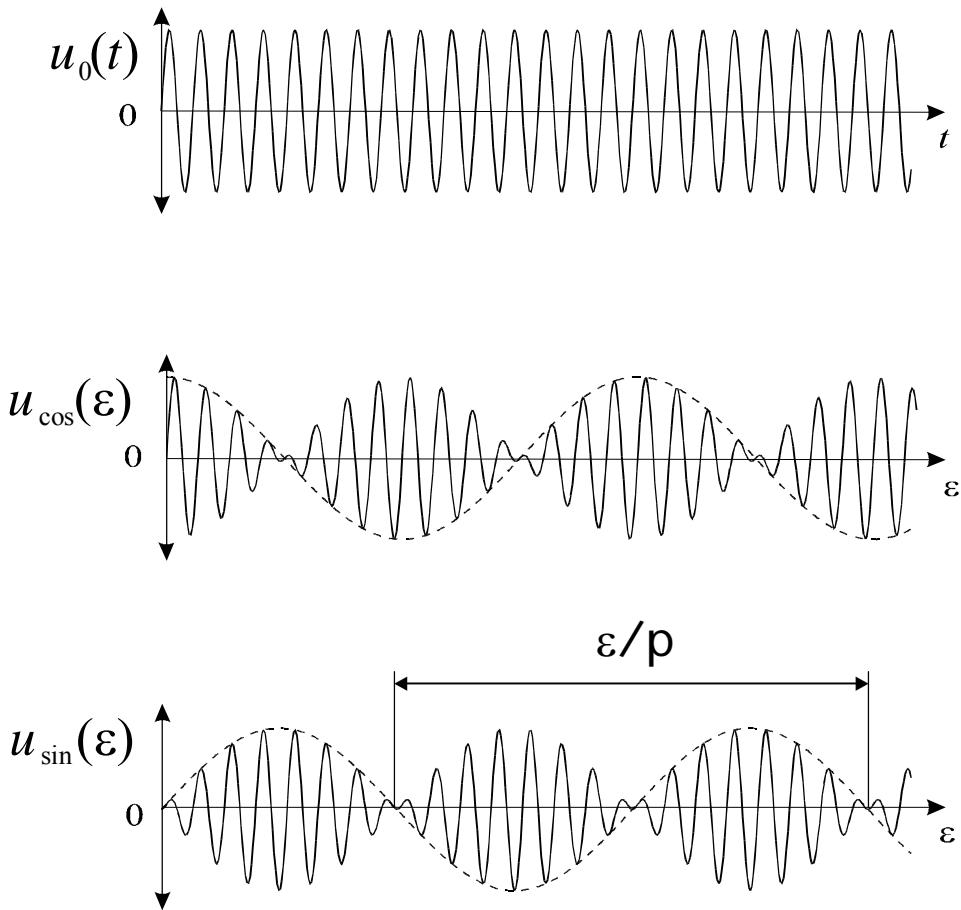
V statorskih navitjih se inducirata napetosti, ki sta odvisni od kota zasuka  $\epsilon$ , kar je razvidno tudi s slike 12.17 (p je število polovih parov resolverja):

$$\begin{aligned} u_{\text{sin}} &= k u_0 \sin(p\epsilon) = k U_0 \sin(\omega t) \sin(p\epsilon) \\ u_{\text{cos}} &= k u_0 \cos(p\epsilon) = k U_0 \sin(\omega t) \cos(p\epsilon). \end{aligned}$$

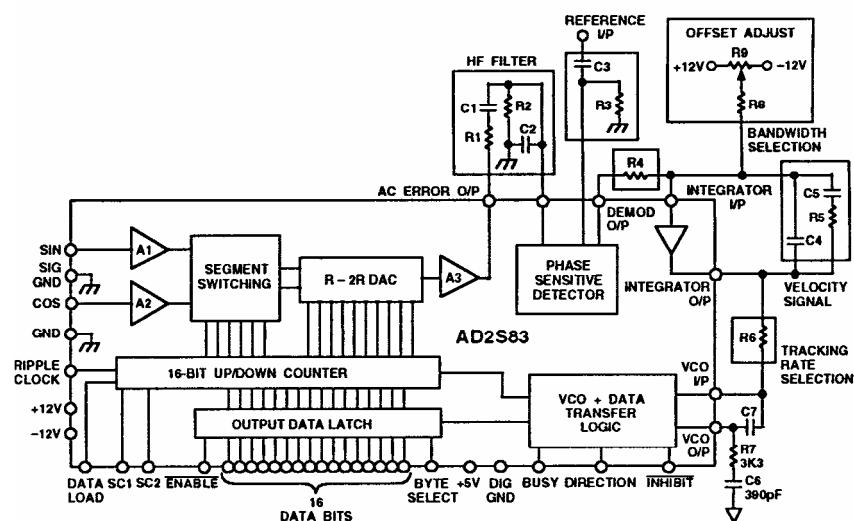
Pri merjenju kota  $\epsilon$  je treba ugotoviti trenutni amplitudi moduliranih signalov (ovojnici na sliki 12.17), ki sta sinusna in kosinusna funkcija kota, nato pa dejanski kot. Za reševanje te zahtevne naloge lahko uporabimo posebna integrirana vezja, kot je AD2S83 firme Analog Devices (slika 12.18). Na ta način se izognemo dodatni aparturni opremi ter prihranimo dragoceni mikroprocesorski čas.

Vezje potrebuje trojno napajanje (+12 V, -12 V, +5 V). Frekvenca referenčnega signala je največ 20 kHz, njegova medtemenska vrednost 8 V (od pozitivne do negativne amplitude), tipična efektivna vrednost moduliranih signalov pa je 2 V.

<sup>3</sup> Resolverji in inkrementalni dajalniki vedno generirajo signala premaknjena za  $90^\circ$ . Dva signala sta potrebna za ugotavljanje smeri vrtenja, kar je razvidno iz zaporedja generiranih signalov.



Slika 12.17: Oblike referenčnega in dveh izhodnih signalov (sinusnega in kosinusnega) iz resolverja ob spremjanju kota zasuka



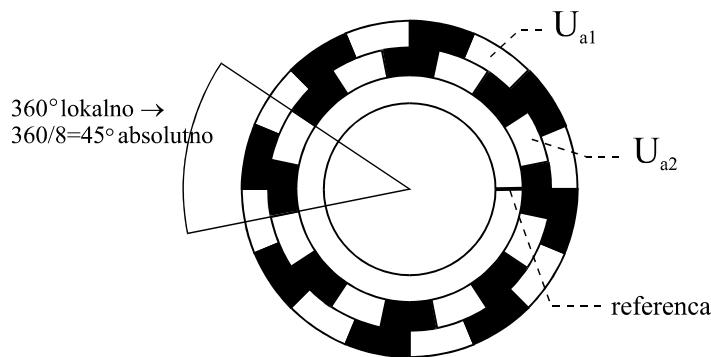
Slika 12.18: Blokovna shema resolverski digitalnega pretvornika AD2S83

Vezje AD2S83 vključuje tudi A/D pretvornik in števec in skrbi za ponazoritev kota v digitalni obliki. Na ta način je možna neposredna povezava z mikroprocesorskim sistemom. S pomočjo dveh binarnih vhodov (SC1 in SC2) lahko izberemo ločljivost 10, 12, 14 ali 16 bitov, kar je odvisno od dinamike posameznega merjenca<sup>4</sup>. Stanje binarnega izhoda DIR (angl. direction - smer) je odvisno od smeri vrtenja.

Integrirano vezje vsebuje tudi signal, ki je proporcionalen kotni hitrosti (izhod INTEGRATOR O/P).

#### 12.4.2 Vezja za zajemanje signalov iz inkrementalnega dajalnika

Inkrementalni dajalnik je najpogosteji merilni člen za zajemanje kota zasuka motorjev [2]. Osnovni deli dajalnika so delno zasenčeni stekleni disk znotraj kolobarjev ter izvora in detektorja svetlobe (slika 12.19). Svetlobno občutljiva senzorja zaznavata prehod svetlobe skozi nezatemnjena področja. Rezultat sta dva za  $90^\circ$  premaknjena vlaka pulzov (slika 12.20). Tretji signal je kratkotrajni pulz, ki se pojavi enkrat na polni obrat. Tudi tukaj je zaporedje pulzov odvisno od smeri vrtenja.



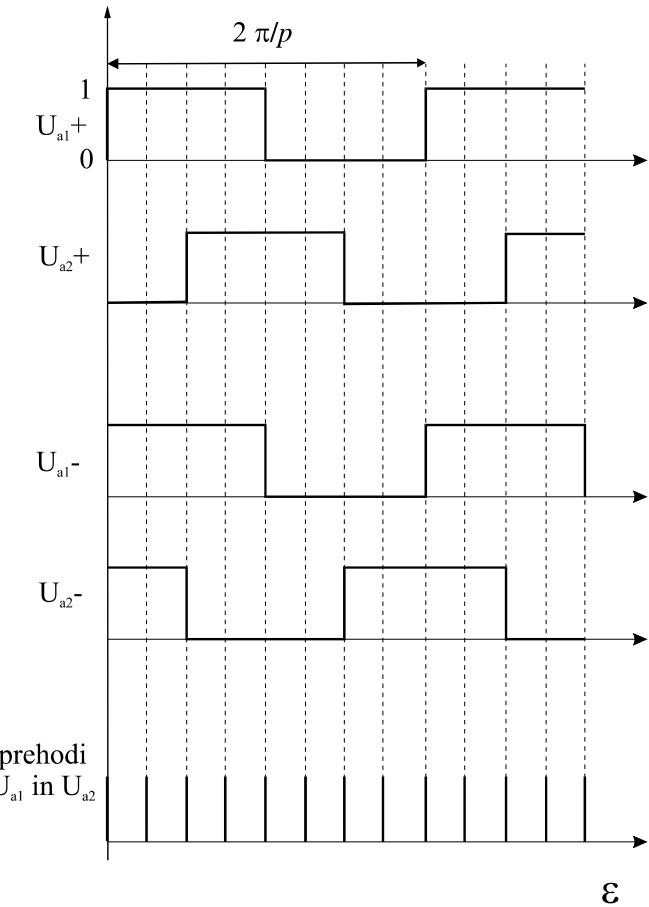
*Slika 12.19: Nameščenost zasenčenih polj na stekleni plošči inkrementalnega dajalnika*

Trenutni kot zasuka ugotavljamo s prištevanjem ali odštevanjem pulzov od nekega referenčnega kota (odvisno od smeri vrtenja). Tak števec mora zadovoljiti naslednje zahteve:

- Mora imeti možnost štetja vseh prehodov na obeh vhodih (slika 12.20). Na ta način se ločljivost dajalnika poveča za štirikrat.
- Ugotavljanje spremembe smeri vrtenja.
- Možnost prednastavitev začetnega stanja.

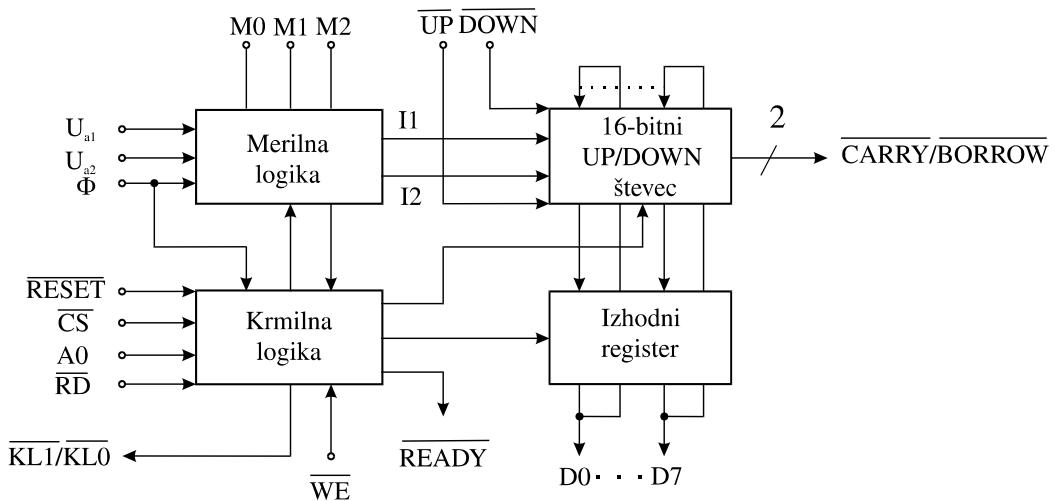
---

<sup>4</sup> Ob ločljivosti 10 bitov je maksimalna možna izmerjena kotna hitrost  $1040 \cdot 2\pi \text{ s}^{-1}$ , pri šestnajstih bitih pa  $16,25 \cdot 2\pi \text{ s}^{-1}$ .



*Slika 12.20: Oblike signalov iz inkrementalnega dajalnika*

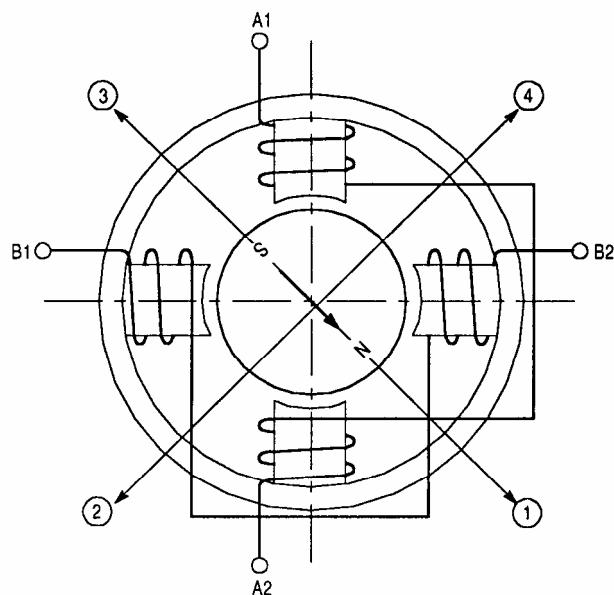
LS2000 firme Texas Instruments je eno starejših vezij za ugotavljanje kota zasuka iz inkrementalnega dajalnika (slika 12.21). Vsi sodobni mikrokrmilniki za krmiljenje in regulacijo elektromotorskih pogonov imajo takšne inteligentne števne module že vgrajene (npr. MC68332, TMS320F240).



*Slika 12.21: Integrirano vezje LS2000*

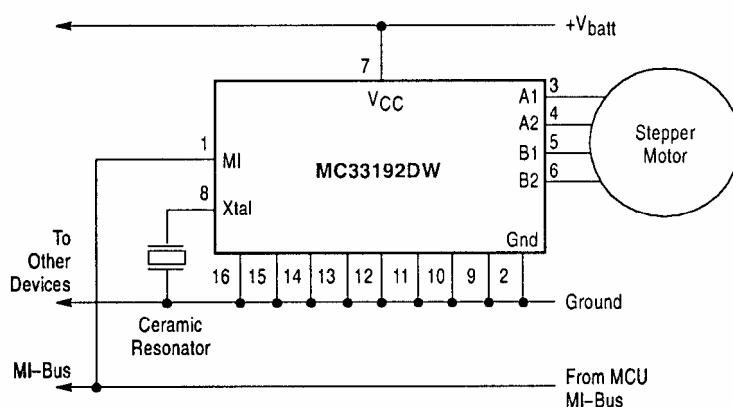
## 12.5 Krmilniki za motorje

Koračne motorje pogosto zasledimo v pogonih manjših moči, kjer sta osnovni zahtevi nizka cena in enostavnost krmiljenja. Primera takšnih aplikacij sta dvigovanje stekel ali pogon brisalnikov v avtomobilih. Značilnost teh motorjev je koračno spremenjanje kota v odvisnosti od vlakov pravokotnih pulzov na navitjih A in B (slika 12.22). Oblika napajalnih napetosti je podobna signalom iz inkrementalnega dajalnika, njihovi nivoji pa so odvisni od načina obratovanja (polni korak ali polovični korak). Večina sodobnih krmilnikov (npr. MC68332 in TMS320F240) že vsebujejo podmodule, ki so sposobni generirati zahtevano obliko pulzov, ostali mikroračunalniki pa lahko uporabijo vezja, kot je MC33192 firme Motorola (slika 12.23).



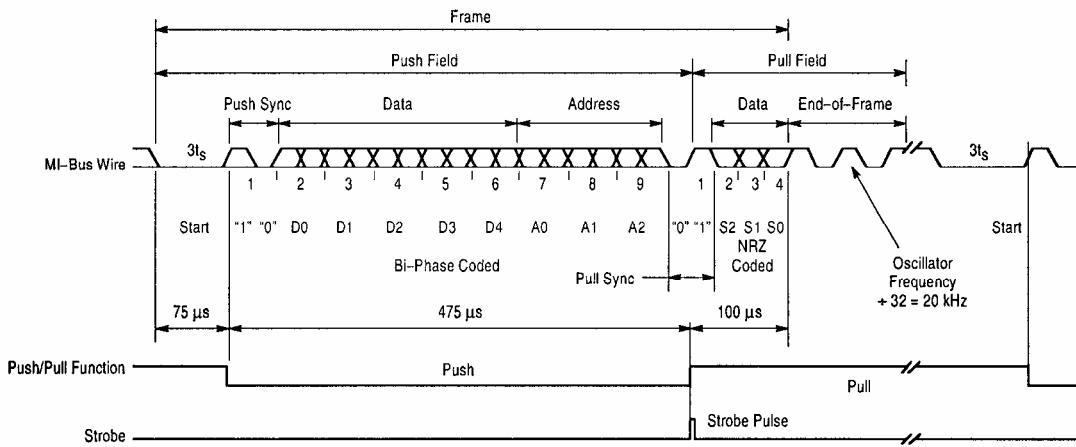
Slika 12.22: Koračni motor

MC33192 je zanimiv tudi kot primer vezja, s katerim mikroračunalnik komunicira po posebnem serijskem vodilu MI-Bus. Na to vodilo je možna priključitev do osmih MC33192, kar ustreza številu motorjev, ki jih lahko krmili en mikroračunalnik. Sponke A1, A2, B1 in B2 priključimo na navitja motorja.



Slika 12.23: MC33192 Krmilnik koračnih motorjev

Serijski podatki za MC33192 prihajajo na vhod MI. Vsak paket je sestavljen iz treh področij (slika 12.24): podatkovni biti (D4-D0), statusni biti (S2-S0) in naslovni biti (A2-A0).

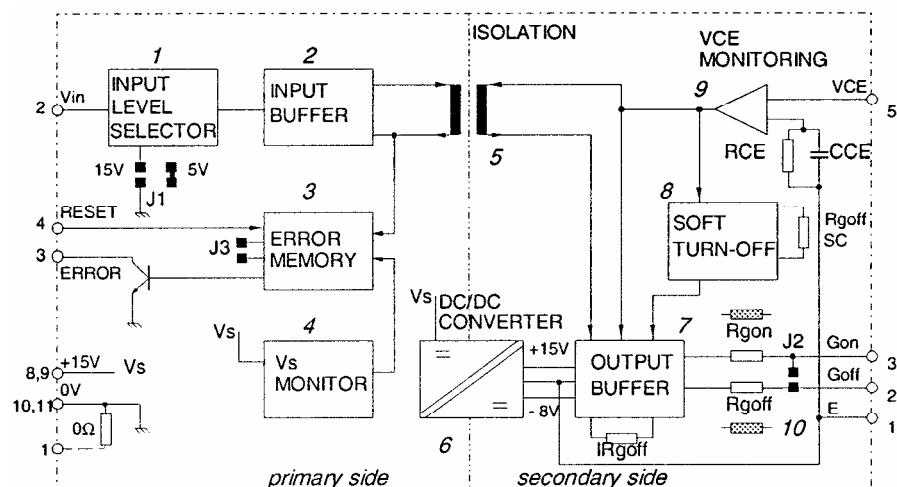


**Slika 12.24:** Časovni diagram vodila MI

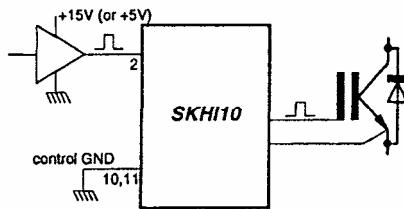
Podatkovni biti določajo sekvence vklapljanja tranzistorjev na gonilniku (angl. driver) izhodnega mostiča za napajanje motorja. Statusni biti določajo trenutno stanje pogona. Tak primer je preobremenitev čipa, ko njegova temperatura preseže 150°C. Z naslovnimi biti izbiramo posamezni čip MC33192 na vodilu MI.

## 12.6 Gonilnik močnostnih IGBT-jev

Vezja močnostne elektronike so pogosto sestavljena iz IGBT tranzistorjev, ki potrebujejo posebne gonilnike. Takšen gonilnik je tudi SKH10 firme Semikron (slika 12.25). Vhodni signal je želeno stanje tranzistorja (vklopljen/izklopljen), ki ga narekuje mikroprocesorski sistem, izhodne signale pa vežemo neposredno na IGBT (slika 12.26).



**Slika 12.25:** Gonilnik za IGBT Semikron SKH10



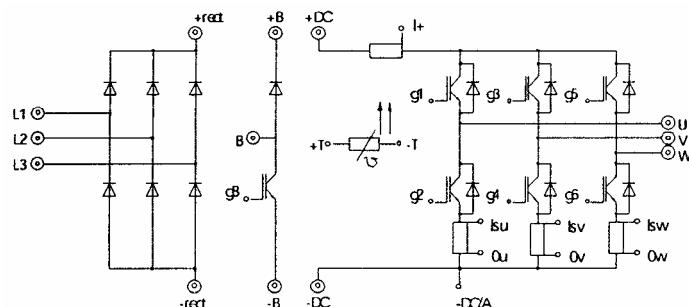
*Slika 12.26: Vezava SKHI10*

### Nekatere lastnosti SKHI10:

- CMOS/TTL (HCMOS) kompatibilni vhodi,
- zaščita pred kratkim stikom,
- galvanska ločitev primar/sekundar preko transformatorjev,
- maksimalna vrednost napajalne napetosti ( $V_S$ ): 18 V,
- maksimalna vrednost vhodnega signala ( $V_{IH}$ ):  $V_S + 0,3$  V,
- maksimalni izhodni tok ( $I_{outPEAK}$ ):  $\pm 8$  A,
- maksimalna merjena napetost kolektor-emitor ( $V_{CE}$ ): 1200 V ali 1700 V,
- zakasnitev izhodnega vklopnega in izklopnegra pulza z ozirom na vhodna pulza: 1,4  $\mu$ s.

## 12.7 Integrirani mostiči

Močnostna mostična vezja so sestavni del večine naprav močnostne elektronike (npr. električni pogoni, resonančni pretvorniki, sistemi za brezprekinitveno napajanje itd.). Razvoj mikroelektronike omogoča uporabo inteligentnih integriranih mostičnih modulov, ki so preirejeni za montažo na tiskana vezja. Primera takšnih integriranih vezij sta seriji SKiiP ali Mini SKiiP firme Semikron (slika 12.27).



*Slika 12.27: Mini SKiiP 2*

Sestavni deli mostiča so homogeni IGBT tranzistorji z protiparalelnimi CAL-diodami. Nekatera vezja vsebujejo tokovne merilnike, ki jih razen za nadtokovno zaščito in zaščito

pred kratkim stikom lahko uporabimo kot povratno informacijo v regulacijski tokovni zanki. Temperaturni senzor je standardni del vsakega vezja, pri nekaterih vezjih pa je dostopna tudi normirana napetost enosmernega vmesnega tokokroga. V primerih, ko delamo v okolju, ki generira motnje, ali pa ko so razdalje med krmilnim (mikroprocesorjem) in močnostnim delom velike, lahko uporabimo inačico vezja z optičnim prenosom krmilnih signalov.

## **K. KRATICE**

|           |   |
|-----------|---|
| An        | - naslovni register MC68332                             |
| BIN       | - dvojiško  |
| CPU       | - centralna procesna enota                              |
| <u>CS</u> | - signal za izbiro čipa (chip select)                   |
| DEC       | - desetiško   |
| Dn        | - podatkovni register MC68332                           |
| DSP       | - digitalni signalni procesor                           |
| HEX       | - šestnajstiško   |
| IMB       | - intermodularno vodilo MC68332                         |
| LSx       | - skrajni levi (bit, byte, beseda)                      |
| MSx       | - skrajnji desni (bit, byte, beseda)                    |
| PC        | - programski števec, osebni računalnik                  |
| PWM       | - pulzno-širinska modulacija                            |
| QSPI      | - podmodul za sinhronsko serijsko komunikacijo MC68332  |
| QSM       | - modul za serijsko komunikacijo MC68332                |
| SCI       | - podmodul za asinhronsko serijsko komunikacijo MC68332 |
| SIM       | - modul za integracijo sistema MC68332                  |
| SP        | - kazalec sklada  |
| SR        | - statusni register                                     |
| TPU       | - časovno-procesna enota MC68332                        |
| UART      | - asinhronski zaporedni vmesnik                         |
| X         | - indeksni register                                     |

## L. LITERATURA

1. Ahmed I.: "Implementation of PID and Deadbeat Controllers with the TMS320 Family", Applicatin Report SPRA083, Digital Signal Products Semiconductor Group, Texas Instrumenst, 1997
2. Ambrožič V.: "Sodobne regulacije pogonov z izmeničnimi stroji", učbenik, FE, 1996
3. Brechmann G. et al.: "Elektrotehniški priročnik", Viharnik, Ljubljana, 1994
4. Cajhen R.: "Regulacije", učbenik, FE, Ljubljana, 1984
5. Dorf R. C. (editor in chief): "The Engineering Handbook", CRC Press, Boca Raton, 1996
6. Gams M.: "Računalniški slovarček", Cankarjeva založba, 1993
7. Gulalo G.: "Worldwide production of electric motors: changes, growth, and driving forces", Details on Digital Control Systems, vol. 1, Issue 1, Texas Instruments Inc, July 1998
8. Howe D. (editor): "The Free On-line Dictionary of Computing, <http://foldoc.doc.ic.ac.uk/>"
9. Kodek D.: "Mikroprocesorski sistemi", BI-TIM d.o.o., Ljubljana, 1993
10. Levine W. S. (editor): "The Control Handbook", CRC Press Inc., Boca Raton, 1996
11. Milman J.: "Microelectronics: Digital and Analog Circuits and Systems", Mc Graw Hill, 1979
12. Moler C.: "Floating Points", Matlab News & Notes, Mathworks, Fall 1996
13. Leonhard W.: "Digitale Signalverarbeitung in der Meß- und Regelungstechnik", B. G. Teubner, Stuttgart, 1989
14. Lilein A. L.: "Digital Processors vs. Universal Microprocessors", ESIEE, Paris, SPRA344, Texas Instruments, 1996
15. "M68332BUG, Debug Monitor User's Manual", Motorola Inc., 1990
16. Ribarić S.: "Arhitektura mikroprocesora", Tehnička knjiga, Zagreb, 1985
17. Ribarić S.: "Naprednije arhitekture mikroprocesora", Školska knjiga, Zagreb, 1990
18. Salčić Z.: "Mikroračunarski sistemi", Svjetlost, Sarajevo, 1984

19. Skupina avtorjev: "The Engineering Handbook", CRC Press, 1996
20. Skupina avtorjev: "Dizionario Enciclopedico Scientifico e Tecnico", McGraw-Hill & Zanichelli, Bologna, 1996
21. Skupina avtorjev: "16 - bit Microprocessor Systems", McGraw-Hill, New York,..., 1982
22. Šilc J., Robič B., Ungerer T.: "Processor Architecture", Springer Verlag, Berlin, Helderberg, 1999
23. Vujčić V.: "Uvod u C jezik", Institut za nuklearne nauke "boris Kidrič", Vinča, Beograd, 1989
24. N. N.: "M68332 User's Manual", Motorola Inc., 1990 in 1993
25. N. N.: "M68300 Family, CPU32 Central Processor Unit, Reference Manual", Motorola Inc., 1990
26. N. N.: "TPU Time Processor Unit, Reference Manual", Motorola Inc., 1990
27. N. N.: Interfacing TMS320C54x DSK-Plus with the TMS28F400, SPRA456, Texas Instruments Europe, Junij 1998
28. N. N. "Sine, Cosine on the TMS320C2xx", Application Report No. BPRA047, Texas Instruments, 1997
29. N. N.: "Field Oriented Control of 3-Phase AC-Motors", Texas Instruments Europe, Literature No. BPRA073, February 1997
30. N. N.: "Implementation of a Speed Field Orientated Control of Three Phase AC Induction Motor Using TMS320F240", Texas Instruments Europe, Literature No. BPRA076, February 1998
31. N. N.: "Data Transmission Design Seminar, Reference Manual", Texas Instruments, 1998
32. N. N.: "TMS320F240 Evaluation Module, Technical Reference", Spectrum Digital Inc., 1998
33. N. N.: "Intertools 68000 Family C Compiler/Assembler: Reference Manual, User's Manual", Intermetrics Microsystem Software, Inc., Cambridge, 1992
34. Self. K: "Memory in megabytes and/or mebibytes", IEEE Spectrum, Avgust 1999

*Spletne strani proizvajalcev mikroprocesorskih komponent:*

<http://www.amd.com>  
<http://www.analogdevices.com>  
<http://www.burr-brown.com>

<http://www.hitachi.com>  
<http://www.maxim-ic.com>  
<http://www.micron.com>  
<http://www.motorola.com>  
<http://www.philips.com>  
<http://www.smi.siemens.com>  
<http://semikron.com>  
<http://www.ti.com>  
<http://www.xeltek.com>

Zgoščenke:

“TMS320C24x, Digital Motor Control”, Texas Instruments, 1999

“TMS320C6xx Digital Signal Processors”, Texas Instruments, 1997

“TMS320 DSP Solutions”, Texas Instruments, 1997