Kazalo

1 Nekaj iz zgodovine računalnikov

Zgodovinskemu razvoju računalnikov bomo sledili skozi tri tehnološka obdobja:

- obdobje mehanike (17., 18. stoletje),
- obdobje elektromehanike (konec 19. in začetek 20. stoletja) in
- obdobje elektronike (1940 $\rightarrow$).

1.1 Obdobje mehanike v Evropi

Za to obdobje so značilni stroji za seštevanje, odštevanje, množenje in deljenje, z malo ali nič možnosti za programiranje. Takim strojem navadno rečemo kalkulatorji.

Wilhelm Schickard (1592 - 1635) profesor matematike (Tubingen) je leta 1623 izdelal (malo znan) stroj za seštevanje, odštevanje in množenje. Edini ohranjen zapis o tej 'uri za računanje' je Schichardovo pismo Johanesu Keplerju. Njegov stroj ni imel neposrednega vpliva na kasnejši razvoj strojev za računanje.

Blaise Pascal (1623 - 1662) je leta 1642 izdelal bolj znan, a manj zmogljiv stroj za seštevanje in odštevanje. Njegov stroj je imel dve skupini s po šestimi zobatimi kolesi. Ena skupina koles je služila kot šestmestni akumulator, druga skupina je služila za vnos števila za prištevanje ali odštevanje. Pogon je bil ročen. Kasneje je Pascal izdelal še več podobnih naprav.

Gotfried Leibniz (1646 - 1716) je leta 1671 zgradil stroj, ki je znal vse štiri osnovne računske operacije $+, -, *, /$. Leibnizove zamisli so imele bistven vpliv na kasnejši razvoj in so dolgo časa doživljale neprekinjen tok izboljšav. Žal takratna tehnologija ni mogla zagotoviti zadostne zanesljivosti teh naprav. Prvi komercialni in resnično uporabni stroji so se pojavili šele v drugi polovici devetnajstega stoletja, ohranili pa so se dokler jih niso sredi tega stoletja izpodrinili elektronski kalkulatorji.

Charles Babbage (1792 - 1871) je praktično vse svoje življenje posvetil gradnji strojev za računanje. Zasnoval je in skušal zgraditi dva stroja, ki sta bila neprimerno naprednejša od predhodnikov, pa tudi naslednikov.

Prvi in preprostejši **diferenčni stroj** je znal seštevati in v povezavi z metodo končih diferenc omogočal izračun vrednosti poljubnega polinoma. Delo se je pričelo 1823 in je trajalo do 1833, ko je bil projekt zaradi pomankanja denarja in nesoglasij opuščen. Babbageovo zamisel pa je uresničil skupaj s sinom bogati švedski tiskar Georg Scheutz. Diferenčni stroj je prvi realiziral zamisel o *avtomatskem izvajanju* zaporedja operacij, kar je praktično osnovna razlika med kalkulatorjem in računalnikom.

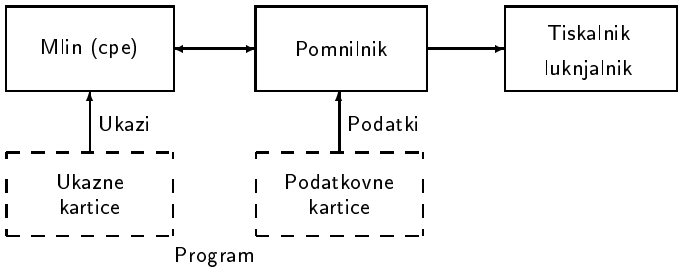
Drugi in naprednejši **analitični stroj** (1834) je bil namenjen za računanje poljubne matematične operacije, žal pa ni bil nikoli do konca zgrajen. Stroj sta sestavljala dva dela: mlin in pomnilnik.

- Mlin je služil za izvajanje aritmetičnih operacij $(+, -, *, /)$. Mlin je po funkciji ustrezal današnji centralni procesni enoti. Seštevanje naj bi trajalo okrog ene sekunde, množenje pod minuto.
- Pomnilnik za shranjevanje podatkov naj bi imel velikost 1000 x 50-mestnih desetiških števil.

Za določanje zaporedja izvajanja operacij je Babbage predvidel luknjane kartice (Jacquardove kartice). Predvidel je:

- Ukazne kartice: operacijske in kombinatorične (za krmiljenje zaporedja izvajanja operacij),
- podatkovne kartice (kje so operandi in kam gredo rezultati),

Rezultati naj bi se izpisovali ali luknjali na kartice.



Pomembno za analitični stroj je, da:

- delovanje vodi program,
- omogoča reševanje splošnih problemov,
- bistven prispevek je možnost *krmiljenja zaporedja izvrševanja operacij.*

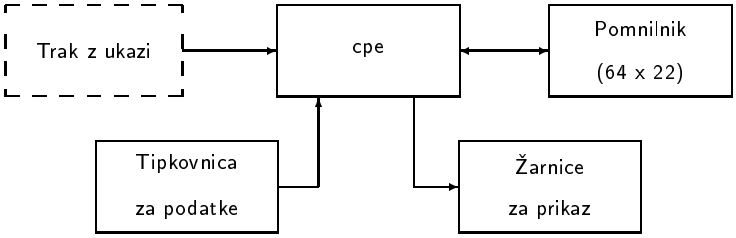
1.2 Obdobje elektromehanike (releji in motorji)

Herman Hollerith (1860 - 1929) gradi naprave za štetje, sortiranje in tabeliranje na osnovi luknjanih kartic. Stroji so bili komercialno uspešni (uporabili so jih pri popisu prebivalstva v ZDA 1890) in leta 1924 Hollerith soustanovi IBM (International Business Machines). Razvojno gledano pa so te naprave skoraj korak nazaj.

Od leta 1930 teče razvoj neodvisno v ZDA in v Nemčiji. V Nemčiji je **Konrad Zuse** (1910 - 1995) leta 1938 zgradil mehanični računalnik **Z1**. Zuse ni vedel za Babbagevo delo. Prvi je uporabil *binarno aritmetiko in aritmetiko s plavajočo vejico.*

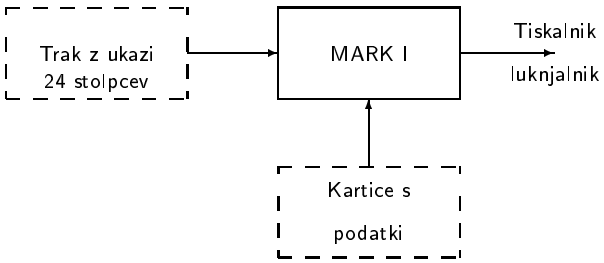
Leta 1941 je Zuse zgradil elektromehanični računalnik Z3, za katerega je danes splošno priznano, da je bil prvi programljiv računalnik za splošne namene. Nekaj zanimivosti tega stroja:

- okrog 2600 telefonskih relejev,
- pomnilnik z 64 x 22 bitnimi besedami (za podatke),
- luknjan trak za ukaze,
- tipkovnico za vnos podatkov,
- žarnice za prikaz,
- ni obvladal programskih skokov.



Zuse ni vedel za delo Američanov v letih 1930, Američani niso vedeli za Zuseja.

Howard Aiken (1900 - 1973) (fizik na Harvardski univerzi) je poznal Babbagevo delo in se je zgledoval po njem. Leta 1937 je bil dan predlog za projekt **MARK I**, 1939 delo začne in 1943 je projekt končan. Za delo je Aiken imel podporo pri IBM-u (Thomas Watson).



MARK I je bil v uporabi na Harvardu od 1944 do 1959. Nekaj zanimivih podatkov:

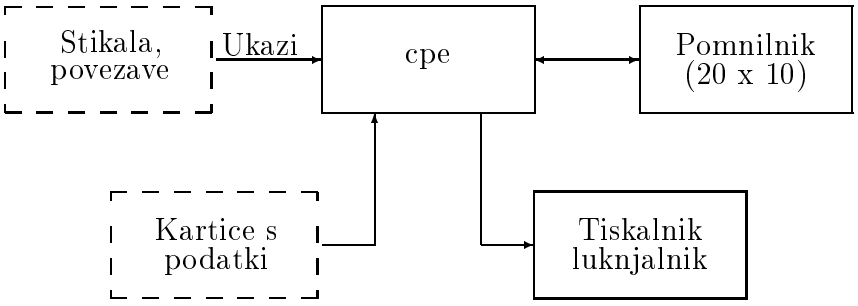
- 15 m dolg, preko 2 m visok in širok,
- desetiška cela števila, desetiška števna kolesa (elektromehanska),
- pomnilnik za 72 23 mestnih desetiških števil,
- ukazi oblike: A1 A2 OP, seštevanje/odštevanje je trajalo 0.3 s, množenje 3 s, deljenje 6 s.
- ni imel pogojnih skokov,
- ukazi na luknjanem traku, podatki na karticah,
- rezultati so se tiskali na tiskalnik ali luknjali na trak.

Kasneje so mu sledili računalniki MARK II, III, IV.

1.3 Obdobje elektronike

John Atanasoff (Iowa State University) je najprej delal na problematiki analognih računalnikov. Zgradil je analogni računalnik z vakumskimi elektronkami za reševanje sistemov linearnih diferencialnih enačb. Leta 1935 se preusmeri in do leta 1938 dela na razvoju logičnih elementov. Leta 1939 se loti gradnje velikega digitalnega računalnika, vendar stroja ne dokonča.

Prvi popolnoma elektronski računalnik za splošne namene zgradijo v Philadelphiji leta 1946 na Moore Scholl of Electrical Engineering, University of Pennsylvania. Dajo mu ime **ENIAC** (Electronic Numerical Integrator And Calculator). Projekt ENIAC sta vodila **John Mauchly, Presper Eckert**, **John von Neumann** pa je bil svetovalec na projektu.



Nekaj zanimivosti o ENIACu:

- 18000 elektronk, 200 kW moči, teža 300 ton.
- Pomnilnik za 20 x 10 mestnih desetiških števil.
- Računanje s števili s fiksno vejico, znal je seštevati, odštevati, množiti, deliti in koreniti.
- Hitrost: 5000 seštevanj ali odštevanj na sekundo.
- Programiralo se ga je s stikali in povezavami (preko 6000 večpoložajnih stikal), ker bi trak upočasnil delovanje.
- Programiranje je bilo izredno težavno. V zvezi z ENIACom se prvič uporablja izraz *programiranje*.

1.4 Računalnik s shranjenim programom

Že med nastajanjem ENIACa je bil po zamisli von Neumanna zasnovan naprednejši računalnik **EDVAC** (Electronic Discrete Variable Computer). Važen razvojni napredek EDVACa je, da se:

- ukazi in podatki nahajajo v istem pomnilniku.

EDVAC dokončajo šele leta 1951. Zato ga prehiti angleški EDSAC (1949) (Maurice Wilkes). Nekaj podatkov o EDVACu:

- glavni pomnilnik 1024 x 16 bitnih besed,
- ni imel programskega števca, naslov naslednjega ukaza je bil dan ekspicitno znotraj tekočega ukaza,
  ukazi so bili oblike: A1 A2 A3 A4 OP, kjer je A4 naslov naslednjega ukaza.
- centralna procesna enota nima programsko dostopnih registrov.

Pridobitve računalnika s shranjenim programom:

- dostop do ukazov in podatkov je hiter,
- možnost spreminjanja programa med izvajanjem,
- možnost krmiljenja programa (programski skoki),
- enotno obravnavanje programov in podatkov, ni razlike med programi in podatki.

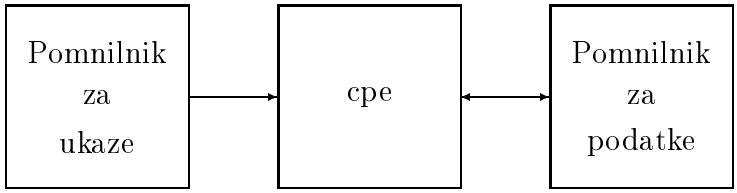
Računalnik EDVAC in kasneje računalnik IAS smemo imeti za predhodnika večine današnjih računalnikov. Računalnik IAS je bil zgrajen pod vodstvom von Neumanna v Princetonu (Institute For Advanced Study) (1952). Značilnosti IAS računalnika:

- prvič je uporabljen programski števec,
- CPE vsebuje več hitrih registrov, od teh dva programerju dostopna registra, ukazi oblike: OP A.
- pomnilnik v obsegu 4096 x 40 bitnih besed, pri čemer ena beseda vsebuje dva ukaza (torej 20 bitov za ukaz) ali predznačeno celo število.

1.5 Harvardski in Princetonski tip računalnika

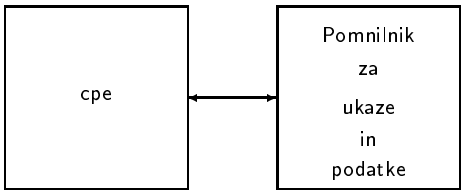
Obstajata dva glavna tipa računalnikov za splošne namene:

- **Harvardski tip** računalnika z enim pomnilnikom za ukaze in enim pomnilnikom za podatke. Pomnilnik ukazov je ločen od pomnilnika podatkov.



Današnji računalniki harvardskega tipa nimajo ločenega pomnilnika za ukaze in podatke (operande), ločene so samo prenosne poti (vodila) med pomnilnikom in procesno enoto. S tem se poveča hitrost prenosa iz/v pomnilnik na račun večjega števila povezav. Da se število povezav ne poveča, pa obstajajo tudi procesne enote, ki imajo sicer 'navznoter harvardsko arhitekturo', 'navzven pa princetonsko'.

- **Princetonski ali von Neumannov tip** računalnika z enim samim pomnilnikom. Pomnilnik hrani ukaze in podatke.



Za von Neumannov računalnik je bistvenega pomena ideja o shranjenem programu, pri čemer se program (ukazi) nahaja v istem pomnilniku kot podatki (operandi). S časom se je pojm von Neumannov računalnik razširil. Danes velja harvardski tip računalnika samo za posebno obliko von Neumannovega računalnika.

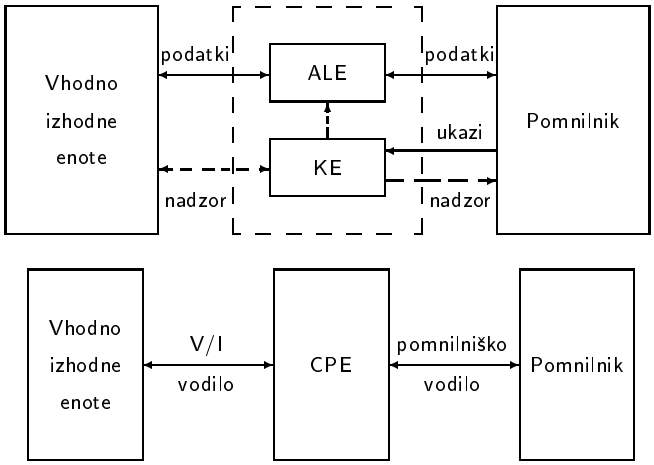
Von Neumannov računalnik je danes sinonim za računalnike tipa (arhitekture) SISD (Single Instruction Single Data Stream).

Oba, harvardski in princetonski tip računalnika sta von Neumannovega tipa.

2 Zgradba računalnika

Večina današnjih računalnikov za splošne namene (General Purpose Computer) je zgrajenih po zamisli Johna von Neumanna iz sredine 1940 let (Burks, Goldstine, Von Neumann). Sestavljajo ga sledeči glavni deli (enote):

- centralna procesna enota **CPE** (ali kar procesor),
  - krmilna enota (KE),
  - aritmetično logična enota (ALE) in
  - registri
- pomnilnik **za ukaze in podatke**,
- vhodno izhodne enote (ali vhodno izhodni podsistem),
- sistem povezav ali vodil.

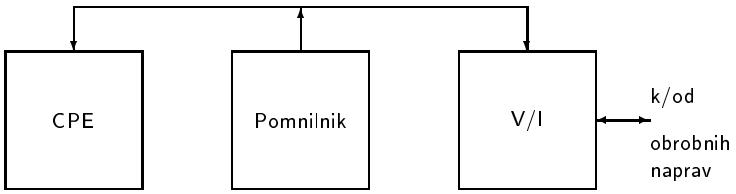


Takim računalnikom pravimo, da so zgrajeni po von Neumannovem vzoru, ali da so von Neumannovi računalniki za razliko od ne-von Neumannovih računalnikov.

Leta 1969 je DEC (Digital Equipment Corporation) v svoji seriji majhnih računalnikov PDP-11 (Programmed Data Processor) prvi združil pomnilniško vodilo z vhodno/izhodnim vodilom v eno samo vodilo. Takšne računalnike danes imenujemo **računalniki s skupnim vodilom**. Večina majhnih računalnikov je danes zgrajenih po tej zamisli. S tem v zvezi govorimo o:

- računalnikih z ločenim v/i vodilom in
- računalnikih s skupnim vodilom.

Računalniki ene ali druge vrste sodijo med računalnike von Neumannovega tipa.



2.1 Splošen opis posameznih delov (enot) računalnika

Pomnilnik je del računalnika, ki hrani programe (ukaze in podatke), vmesne in končne rezultate. Pravimo mu tudi **glavni** pomnilnik za razliko od medpomnilnikov, predpomnilnikov in zunanjih pomnilnikov.

Centralna procesna enota (CPE) je osrednji del računalnika. Njena temeljna naloga je, da iz pomnilnika jemlje ukaze in jih izvršuje.

- **Aritmetično logična enota (ALE)** je del cpe, ki opravlja nekatere aritmetične (+,-,*,/) in nekatere logične operacije (in, ali, ne, pomik).

- **Kontrolna enota (KE)** je del cpe, ki iz pomnilnika prevzema (fetch) ukaze, jih razpoznava ali dekodira in časovno vodi izvršitev ustrezne operacije (execute).

Vhodno izhodne (v/i) enote služijo za prenos podatkov med zunanjim svetom in računalnikom. Povezujejo vhodne in izhodne naprave na računalnik. V/i naprave skrbijo za pretvarjanje informacije iz oblike, ki je primerna za obdelavo na računalniku, v obliko, ki je primerna za človeka ali kakšno drugo napravo.

Vodila (Bus) povezujejo dele računalnika med seboj. Vodila sestavlja množica povezav s svojimi fizičnimi, električnimi in logičnimi lastnostmi.

Te sestavne dele imajo von Neumannovi in ne-von Neumannovi računalniki. Kar računalnik označuje za von Neumannov je (Baron, Higbie):

- delovanje vodi program - njegovo delovanje je popolnoma določeno z zaporedjem ukazov,
- zaporedno izvajanje ukazov - ali pa je vsaj videti tako,
- glavni pomnilnik hrani ukaze in podatke,
- obstaja eno pot med CPE in pomnilnikom - ali pa je vsaj videti tako.

2.1.1 Arhitektura in organizacija računalnika

Arhitektura računalnika je ustroj računalnika, kot ga vidi programer, ki programira v strojnem (ali zbirnem) jeziku (Myers). Ko govorimo o arhitekturi računalnika, obravnavamo funkcionalno zgradbo računalnika neodvisno od načina njegove realizacije. Arhitektura se nanaša na:

- programski model cpe (registre, ki jih vidi programer), nabor ukazov, načine naslavljanja, vhodno izhodne prenose, i.t.d..

Organizacija računalnika je logična zgradba in so lastnosti sestavnih delov računalnika ter njihovih medsebojnih povezav, s katero je realizirana arhitektura. Organizacija se nanaša na lastnosti računalnika, ki so programerju prikrite in so zanj tudi nepomembne:

- nadzorni signali (beri/piši, ...), način izvajanja ukazov, mikroprogramiranje ukazov ali ožičena logika, organizacija pomnilnika.

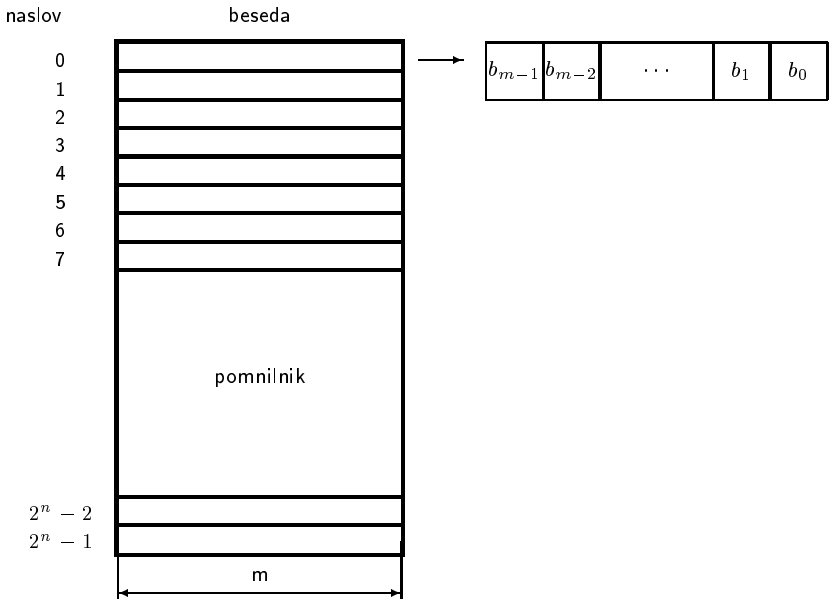
Namesto organizacija danes več uporabljamo izraz **arhitektura sestavnih delov**, npr. arhitektura mikroprocesorjev. Izraz organizacija izginja, izraz arhitektura pa se je razširil.

Obravnavanje **Arhitekture** računalnika pomeni obravnavanje računalnika z vidika, ki je neodvisen od načina njegove realizacije.

2.2 Pomnilnik

Glavni pomnilnik je tisti pomnilnik, do katerega ima CPE neposreden (naključni) dostop. S stališča CPE je pomnilnik linearno (enodimenzionalno) zaporedje pomnilniških besed (Memory Word).

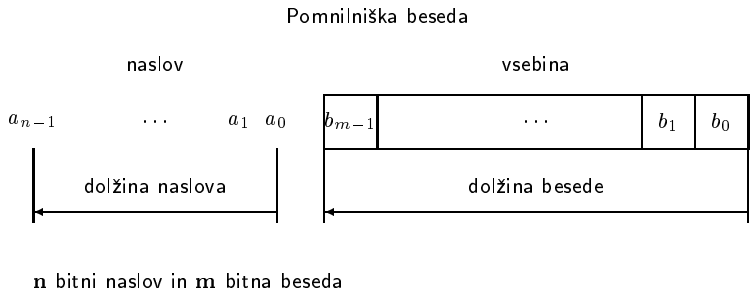
Vsaka beseda (lokacija ali celica) ima svoj naslov. Naslov enolično določa pomnilniško besedo. Številu bitov (**n**) v naslovu pravimo **dolžina naslova**. Od dolžine naslova je odvisna velikost (2^n) naslovnega področja (fizično naslovno področje).



Beseda obsega več **bitov** - bitnih celic. Številu bitnih celic v besedi (**m**) pravimo **dolžina pomnilniške besede**. Najobičajnejša dolžina besede danes je 8 bitov (Bajt ali tudi zlog).

8 bitov = 1 bajt (od leta 1964)
4 biti = 1 nibble ali pol bajta
16 bitov = 2 bajta

Z izrazom beseda včasih označujemo dvojni bajt (16 bitov). Za nas bo **pomnilniška beseda** najmanjša naslovljiva pomnilniška enota.



n bitni naslov in **m** bitna beseda

Glavne lastnosti pomnilnika von Neumannovega računalnika:

- en sam pomnilnik, pomnilnik je enodimenzionalen in organiziran v besede, ni razlike med ukazi in podatki (operandi), pomen podatka ni sestavni del podatka (operanda).

2.3 Pomnilnik in CPE

CPE komunicira s pomnilnikom po (pomnilniškem) vodilu. Vodilo delimo na:

- naslovno vodilo, podatkovno vodilo in kontrolno vodilo.

Vsako od vodil obsega večje število povezav. Številu paralelnih povezav rečemo širina vodila. Vendar, kadar govorimo o širini vodila, imamo običajno v mislih širino podatkovnega vodila.

2.3.1 Operacije CPE s pomnilnikom

Dogajanja na vodilu vodi CPE. Na njeno pobudo se opravi prenos informacije v pomnilnik ali iz pomnilnika. Času, ki je potreben za izvedbo ene take operacije, rečemo pomnilniški cikel, operaciji sami pa dostop do pomnilnika.

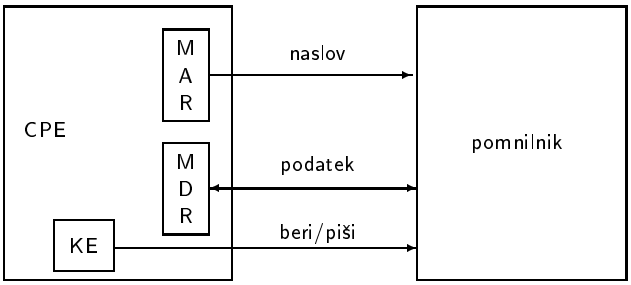
Za **dostop** do **pomnilnika** CPE uporablja:

- **MAR**: Naslovni register pomnilnika (Memory Address Register), ki drži naslov naslovljene pomnilniške besede,
- **MDR**: Podatkovni register pomnilnika (Memory Data Register), ki drži vsebino, ki gre v pomnilnik ali iz pomnilnika.
- Signal Beri/Piši. Beri: prenos iz pomnilnika v CPE. Piši: prenos iz CPE v pomnilnik.

Operacija s pomnilnikom se odvija takole:

- CPE postavi naslov pomnilniške besede na naslovno vodilo (naslovi pomnilnik),
- izbere smer prenosa k ali od CPE (Beri/Piši) (Read/Write). Skratka, postavi ustrezne krmilne signale na kontrolnem vodilu,
- poskrbi, da se podatek prenese po podatkovnem vodilu.

Dostop do pomnilnika se izvede časovno nedeljivo (ko se enkrat začne se izvede do konca brez prekinitve). To je bistvenega pomena v sistemih, kjer je na vodilo vezanih več aktivnih elementov (npr. procesnih enot), ki občasno lahko prevzamejo nadzor nad vodilom.



2.3.2 Hitrost dostopa in čas dostopa

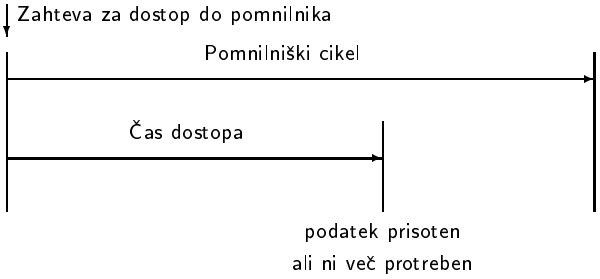
Za glavni pomnilnik je značilen naključni dostop - čas dostopa do podatka je neodvisen od njegovega naslova in je vedno enak, ne glede na zaporedje naslavljanj.

Čas dostopa je tisti čas, ki preteče od trenutka, ko je dana zahteva za prenos do trenutka, ko je informacija prisotna ali ni več potrebna. Čas dostopa za branje je približno enak času dostopa za pisanje.

Hitrost dostopa je enaka številu prenešenih pomnilniških besed na časovno enoto (sekundo).

$$\text{hitrost dostopa} = \frac{1}{\text{pomnilniški cikel}} \neq \frac{1}{\text{čas dostopa}}$$

Poleg časa dostopa je potreben še nek obnovitveni čas, predno je možen naslednji pomnilniški cikel (dostop do pomnilnika).



2.3.3 Delitev pomnilnikov

Delitev pomnilnikov glede na obstojnost:

- destruktivni: z branjem se informaciju izgubi,
- nedestruktivni: z branjem se informacija ne izgubi,
- dinamični: Potrebno občasno osveževanje vsebine (na nekaj ms),
- statični: ni potrebno osveževanje,

- ohranljivi: z odklopom napajanja se inoformacija ne izgubi,
- neohranljivi: z odklopom se informacija izgubi,

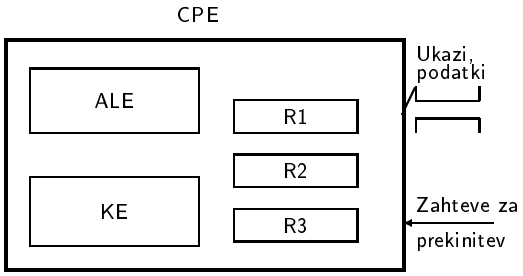
Delitev glede na spremenljivost informacije:

- bralni
  - ROM: (Read Only Memory) vsebina je določena z izdelavo
  - PROM: (Programable ROM) vsebino se da določiti z vpisom (programiranjem).
  - EPROM: (Eraseble ROM) vsebino se določi z vpisom (programiranjem); možno jo je brisati z osvetlitvijo z ultravijolično svetlobo.
  - EEROM: (Electrically Eraseble ROM), vsebino se da vpisati (programirati) in brisati električno.
- bralno/pisalni (RAM).

2.4 Centralna procesna enota - CPE

2.4.1 Osnovna zgradba CPE

Centralno procesno enoto sestavljajo v glavnem kontrolna enota, aritmetično logična enota in registri.



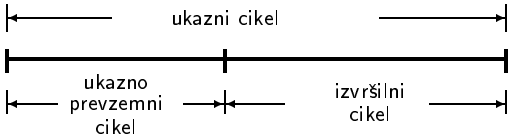
Centralna procesna enota časovno vodi potek vseh dogajanj v računalniku:

- prevzema ukaze iz pomnilnika, jih dekodira in časovno vodi izvršitev operacije, ki jo zahteva ukaz,
- prenaša podatke v/iz pomnilnika ter od/k zunanjim napravam,
- se odziva na zunanje zahteve - prekinitve,

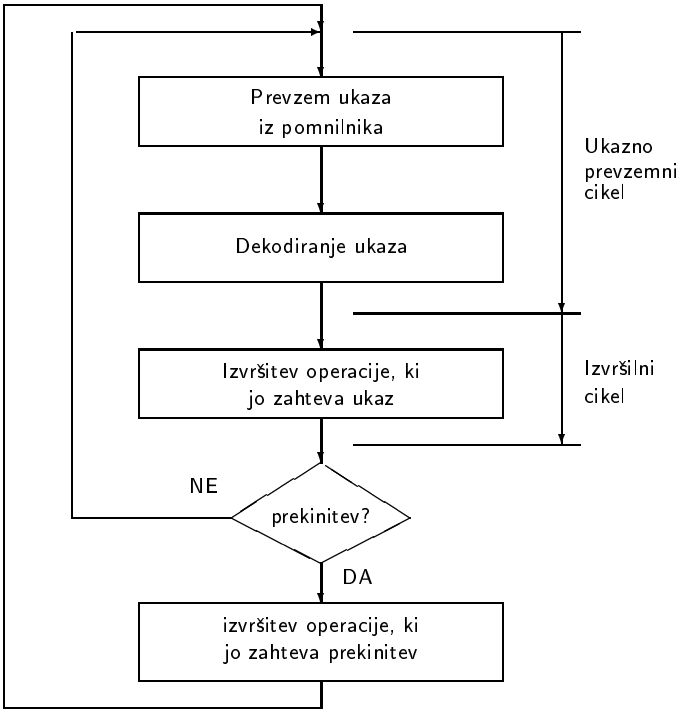
2.4.2 Osnove delovanja CPE

Glavna naloga CPE je izvajanje ukazov. CPE izvaja ukaz za ukazom. Ko konča tekoči ukaz, začne izvajati naslednjega. Času, ki je potreben za obdelavo enega ukaza, pravimo ukazni cikel (Instruction Cyle). Ukazni cikel je sestavljen iz dveh delov:

- ukazno prevzemnega cikla (Fetch) - branje ukaza iz pomnilnika in njegovo dekodiranje,
- izvršilnega cikla (Execute) - priprava vseh operandov in izvedba zahtevane operacije.



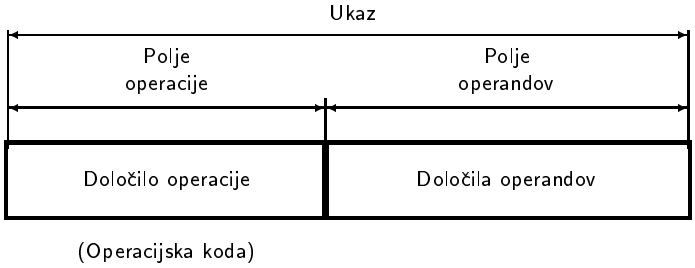
2.4.3 Opis osnovnega delovanja CPE z diagramom poteka



Zahteva za prekinitev CPE pride od zunaj (običajno od zunanje naprave). Videti je, kot da bi CPE prevzela ukaz od zunaj, namesto iz pomnilnika. Zahteva za prekinitev se pojavi asinhrono z izvajanjem ukazov, lahko kadarkoli med izvajanjem ukaza. Vendar se tekoči ukaz **vedno** dokonča, šele nato se streže zahtevi za prekinitev. Izvajanje ukaza je časovno **nedeljivo** (ali pa se prekine le izjemoma, vendar ne kot posledica prekinitve).

2.4.4 Ukazi

Vsak ukaz vsebuje določilo operacije, ki naj se opravi in določila operandov, ki sodelujejo v operaciji. Z obliko ukaza je natančno določeno, koliko in kateri biti pomenijo operacijo, ki naj se izvrši, koliko in kateri biti določajo operande, ki sodelujejo pri operaciji, pa tudi na kakšen način so določeni operandi, s tem pa je določeno tudi, koliko bitov obsega ukaz. Ukaz sestavljata torej dva dela: **določilo operacije** (Operacijska koda) in **določila operandov**.



2.4.5 Možnosti pri izbiri oblike ukazov

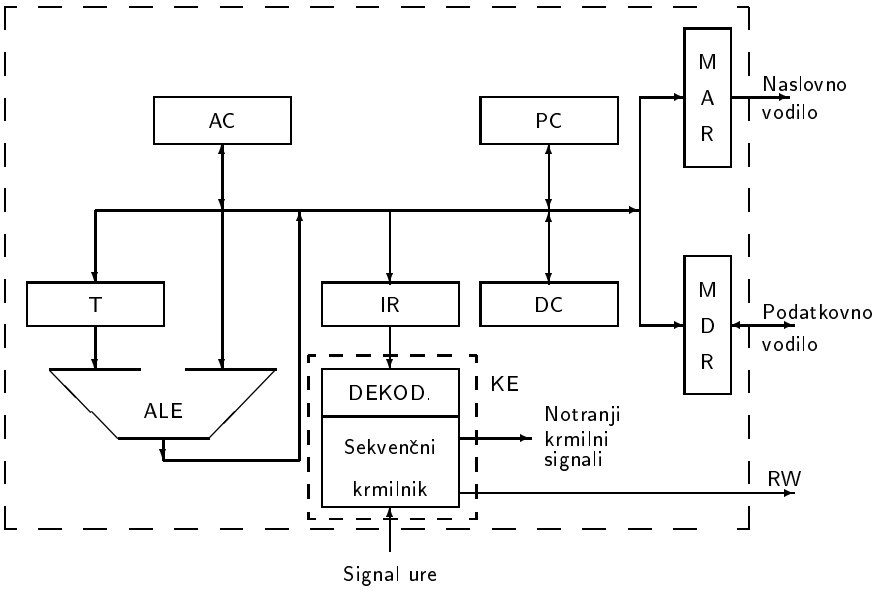
- Vsi ukazi CPE so enake dolžine,
- Ukazi CPE imajo različno dolžino.

- En ukaz - ena pomnilniška beseda (hitro - samo en dostop do pomnilnika),
- En ukaz - več pomnilniških besed (počasnejše - več dostopov do pomnilnika).

Tipično: en ukaz = ena ali več pomnilniških besed in ukazi imajo različne dolžine.

2.5 Izvajanje ukazov CPE

Osnovno delovanje CPE najlažje razložimo na cetralni procesni enoti, ki si jo enostavno izmislimo.

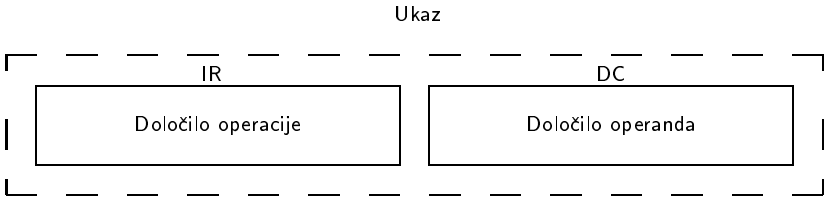


IR	Ukazni register
DC	Operandi register
PC	Programski števec
AC	Akumulator)
ALE	Aritmetično logična enota
KE	Kontrolna enota
T	Začasni register
MAR	Naslovni register pomnilnika
MDR	Podatkovni register pomnilnika

- Programski števec (PC) vsebuje naslov naslednjega ukaza.
- Akumulator (AC) sodeluje pri večini logičnih in aritmetičnih operacij.

- Aritmetično logična enota opravlja aritmetične in logične operacije.
- Začasni register hrani drugi operand pri operacijah z dvema operandoma.
- Ukazni register (IR) hrani tekoči ukaz - za naš primer samo določilo operacije.
- Operandni register (DC) hrani določilo operanda.
- Delovanje CPE časovno pogojuje urin signal (Clock).
- Signal RW določa smer prenosa podatkov po podatkovnem vodilu.

Denimo, da so vsi ukazi CPE enake dolžine in da obsegajo dve pomnilniški besedi, prvo besedo za določilo operacije in drugo za določilo operanda. Tekoči ukaz se iz pomnilnika prevzame v ukazni register in v operandni register.



Poglejmo potek izvajanja ukaza:

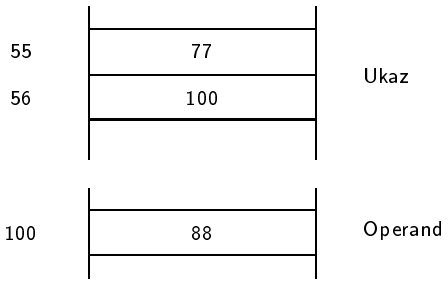
Napolni akumulator z vsebino pomnilniške besede z naslovom 100. Simbolično zapišemo to operacijo običajno takole:

$$AC \leftarrow (100) \quad \text{in} \quad PC \leftarrow PC + 2$$

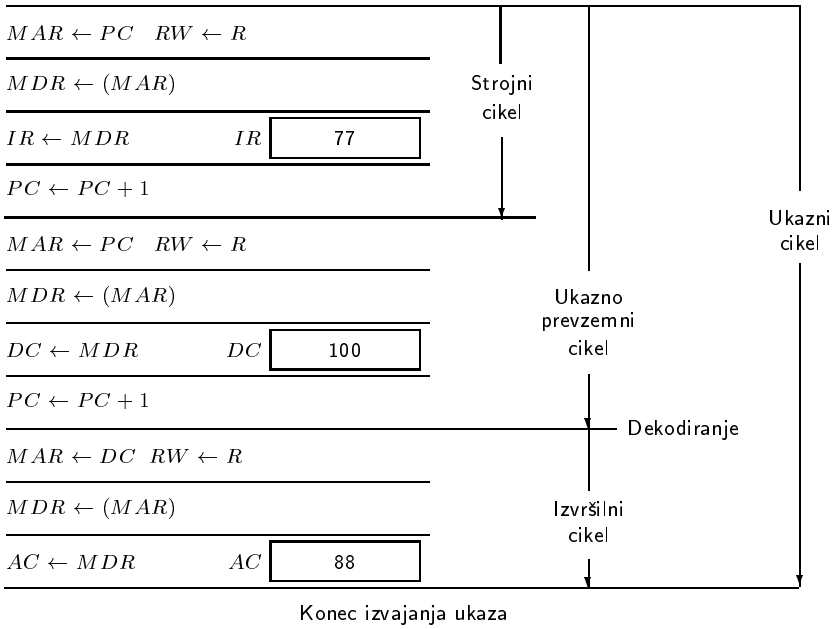
kjer ( ) pomeni vsebino besede z danim naslovom. S črkovno kodo (v zbirnem jeziku) bi bil tak ukaz lahko zapisan tudi takole:

$$LOAD \quad 100$$

Izmislimo si, da je koda operacije enaka 77. Naj bosta naslova pomnilniških besed, ki hranita ukaz, enaka 55 in 56. Naj bo vsebina pomnilniške besede z naslovom 100 enaka 88.

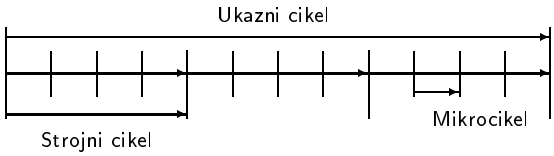


Zapišimo zaporedje korakov za izvedbo tega ukaza. Tik pred začetkom izvajanja ukaza je v programskem števcu prvi naslov ukaza, torej PC = 55.



Pomni: ko je ukazno prevzemni cikel končan, vsebuje programski števec že naslov naslednjega ukaza.

Izvedba ukaza je sestavljena iz zaporedja enostavnejših operacij - mikrooperacij. Čas za izvedbo ene mikrooperacije imenujemo **mikrocikel**. Več mikrociklov sestavlja strojni cikel. V splošnem je en ukazni cikel sestavljen iz več strojnih ciklov (Machine cycles), strojni cikli pa iz več mikrociklov. Ukazno prevzemni cikel in/ali izvršilni cikel obsegata enega ali več strojnih ciklov. Ni nujno, da bi bili vsi strojni cikli enako dolgi.



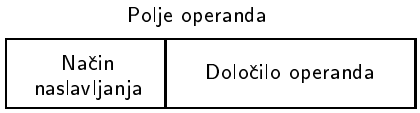
Izvedba mikrooperacij je naloga kontrolne enote. Glede na izvedbo mikrooperacij, ločimo:

- kontrolne enote z ožičeno logiko - operacije so realizirane z logičnimi vezji,
- kontrolne enote z **mikroprogramom**, operacije so realizirane z mikroprogramom v mikroprogramskem pomnilniku, in sicer:
  - mikroprogramirane (nabora ukazov se ne da spreminjati),
  - mikroprogramljive (nabor ukazov se da spremeniti).

Navodilo za izvedbo mikrooperacije imenujemo **mikroukaz**. Zaporedje mikroukazov sestavlja mikroprogram. Ukazi procesne enote so realizirani z mikroprogrami. Zamisel mikroprogramiranja je prvi predlagal Maurice Wilkes leta 1951.

2.6 Načini naslavljanja

Vsak ukaz vsebuje polje, ki določa operacijo in polje, ki določa operande. Operand v ukazu je popolnoma določen z določilom operanda in **načinom**, kako priti do operanda.



Način ali pravilo kako CPE pride do operanda, imenujemo **način naslavljanja**. Zmogljivost CPE je v veliki meri odvisna tudi od možnih načinov naslavljanja.

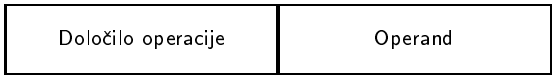
V zvezi z načini naslavljanja govorimo o efektivnem ali **dejanskem** naslovu operanda (Effective Address). Dejanski naslov operanda je naslov operanda, na katerega se nanaša ukaz.

Strogo vzeto, je v polju za določilo operacije samo določilo operacije brez določila operandov ali načina naslavljanja. Vendar se večkrat v to polje 'skrije' tudi informacija o operandih in načinih naslavljanja. To je, operand in/ali način naslavljanja je dan implicitno.

2.6.1 Takojšnje naslavljanje (Imidiate Addressing)

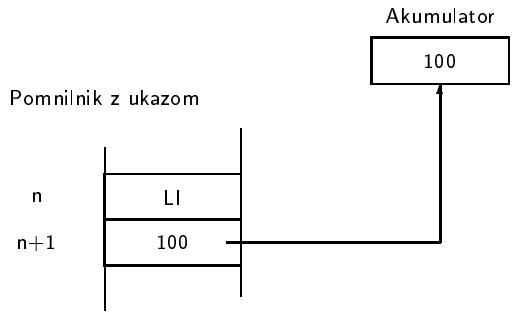
V polju operanda je kar operand.

Takojšne način naslavljanja je primeren za podajanje konstant. Je hiter in enostaven.



Primer: *Napolni akumulator z vrednostjo 100.*

Naj bo črkovna koda operacije recimo **LI**. (Operaciji pridružimo tudi način naslavljanja). Naj ukaz obsega dve pomnilniški besedi.

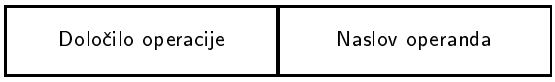


Dolžina polja operanda določa velikost operanda. Pri naboru ukazov z različno dolgimi ukazi srečamo ukaze z različno dolgimi (eno, dvobajtnimi, i.t.d.) operandi.

2.6.2 Neposredno (direktno) naslavljanje (Direct Addressing)

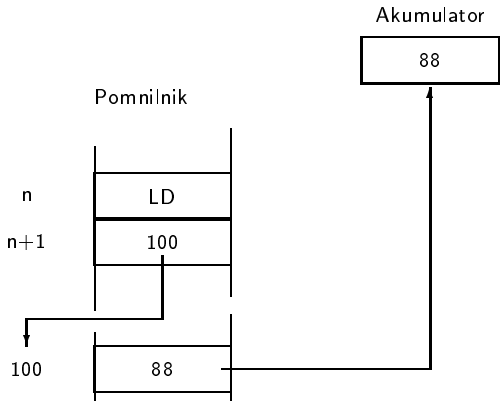
V polju operanda je naslov operanda.

Zahteva dodaten dostop do pomnilnika. Dolžina polja operanda določa velikost naslovnega področja, ki je direktno naslovljivo. Vrednost operanda je spremenljiva - operand se spremeni ne da bi se spremenil ukaz (naslov operanda ostane vedno enak) - tipično za spremenljivke.



Primer: *Napolni akumulator z vsebino pomnilniške besede z naslovom 100.*

Naj bo črkovna koda operacij (skupaj z načinom naslavljanja) **LD**. Naj bo vsebina pomnilniške besede z naslovom 100 enaka 88.

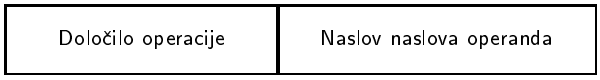


Poseben primer neposrednega naslavljanja je registersko neposredno naslavljanje. Tedaj je v ukazu določilo ('naslov') registra CPE, ki pove, v katerem od registrov je operand.

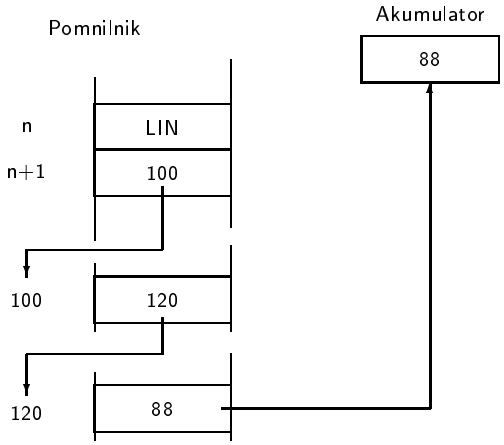
2.6.3 Posredno naslavljanje (Indirect Addressing)

Polje operanda vsebuje naslov naslova operanda.

Zahteva še en dostop več do pomnilnika in je s tega stališča počasnejše. Ta način naslavljanja je pogojila potreba po računanju z naslovi v zvezi z zahtevnejšimi podatkovnimi strukturami (povezani seznami, vrste, ...), kazalci, prenos parametrov med programom in podprogramom.



Primer: *Napolni akumulator z vsebino pomnilniške besede, katere naslov je v pomnilniški besedi z naslovom 100.* Naj bo črkovna koda operacije **LIN**.

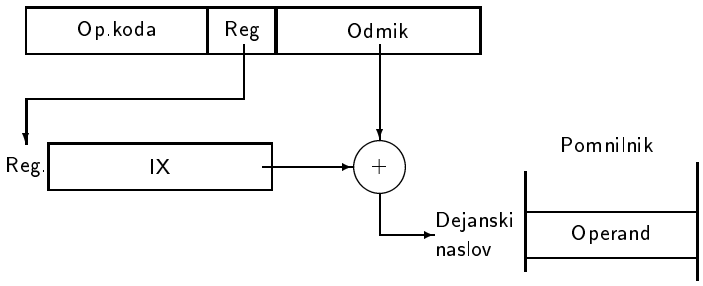


Če je naslov operanda v registru, govorimo o **registerskem posrednem** naslavljanju. V polju operanda je tedaj določilo registra CPE, ki vsebuje naslov operanda. Ker je naslov operanda v registru, je potreben samo še en dostop do pomnilnika. Zato je ta način naslavljanja hiter in gibčen - naslov operanda je možno hitro spremeniti. Najbolj znana primera registerskega posrednega naslavljanja sta:

- indeksno naslavljanje (z indeksnim registrom IX) in
- bazno naslavljanje (z baznim registrom BX).

V obeh primerih je dejanski naslov določen takole:

$$EA = \begin{matrix} IX \\ BX \end{matrix} + Odmik$$



Pri indeksnem naslavljanju (pri baznem pa ne) je možno avtomatsko indeksiranje (zvečanje ali zmanjšanje vsebine registra pred ali po operaciji) in skaliranje naslova. Nadalje, 'odmik' v polju operanda je poln naslov (v obsegu naslovnega področja).

Bazno naslavljanje je podobno, a ne identično, indeksnemu naslavljanju. Pri baznem naslavljanju sodeluje pri določanju dejanskega naslova bazni register. Bistvena razlika

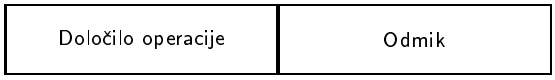
je v tem, da bazni register vsebuje prvi naslov (običajno nekega manjšega) naslovnega področja glavnega pomnilnika, ki je "poln" naslov. Bazni register ne omogoča skaliranja, dekrementiranja ali inkrementiranja.

Proizvajalci (manjših) mikroprocesorjev izraza bazno in indeksno naslavljanje ne uporabljajo dosledno.

2.6.4 Relativno naslavljanje (Relative Addressing) glede na programski števec

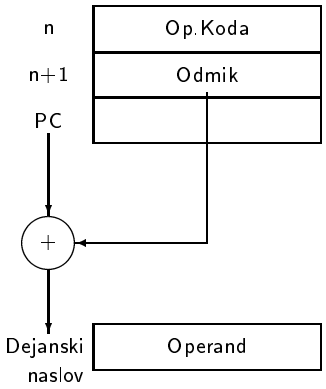
Pri relativnem načinu naslavljanja določata dejanski naslov operanda **vsebina programskega števca** in **vsebina v polju operanda** ukaza (odmik).

Tudi ta način bi lahko šteli med (registrsko) posredne načine naslavljanja.



Relativno naslavljanje je ena izmed možnosti za doseganje položajne neodvisnosti programov (Position Independent Code - PIC). Položajno neodvisen program je tak program, ki se pravilno izvaja ne glede na njegovo namestitev (položaj) v pomnilniku. Torej, če program v pomnilniku premaknemo ne da bi ga kakorkoli drugače spremenili (premestimo drugam), in se program pravilno izvaja naprej, je program položajno neodvisen.

Položajno neodvisnost dosežemo, če se vsi dejanski (efektivni) naslovi v programu spremenjijo tako, da upoštevajo premaknjeni - spremenjeni položaj programa.



Literatura

[1] Kodek D., *Arhitektura računalniških sistemov*, BiTim, Ljubljana 1994.

[2] Kraft G.D., Toy W.N., *Mini/Microcomputer Hardware Design*, Prentice-Hall, 1979.

[3] Baron R.J., Higbie L., *Computer Architecture*, Addison Wesley, 1992.

[4] Derek de Solla Price, "Calculating Machines", IEEE Micro, Feb. 1984, pp.23-52.

3 Motorola 6800

Leta 1974 je Motorola dala na tržišče osembitni mikroprocesor MC6800, ki so mu v kasnejših letih sledila druga z njim združljiva integrirana vezja (vhodno izhodni vmesniki in krmilniki, pomnilniki) visoke integracije. Tem vezjem s skupnim imenom pravimo **mikroračunalniška družina**, ker so med seboj združljiva in jih je možno dokaj enostavno povezovati skupaj.

MC6800 ni bil prvi mikroprocesor. Razvoj prvega mikroprocesorja (4004) je vodil Marcian E. (Ted) Hoff, ki se je leta 1969 zaposlil pri Intel-u kot dvanajsti zaposleni. Skupaj s sodelavci (Federico Faggin, Stan Mazor, Masatoshi Shima) je leta 1971 razvil prvi mikroprocesor z oznako 4004. Mikroprocesorju MC6800 "konkurenčni" mikroprocesor Intel 8080 je zasnoval Masatoši Šima, na tržišču pa je bil nekaj pred njim. (Masatoši Šima, ki je kasneje soustanovil firmo Zilog, je leta 1976 razvil tudi Z80 in Z80000).

Družini osembitnih mikroprocesorjev in vmesnikov M6800 je leta 1979 sledila zmogljivejša družina 16-bitnih in 32-bitnih mikroprocesorjev in vmesnikov in krmilnikov M68000. Ta družina je ohranila koncepte družine M6800 in omogočala celo združevanje z njo, kar je bilo v tistem času pomembno.

Ko začnemo spoznavati nov (mikro)procesor, smo predvsem pozorni na:

- njegove arhitekturne lastnosti: kakšen je programerjev model (izgled mikroprocesorja s stališča programerja), kakšen je nabor ukazov (koliko jih je in kakšni so), kakšni so načini naslavljanja,
- kakšen je sistem prekinitev (ali v splošnem sistem izjem),
- na sponke oziroma signale (signal ure, naslovne sponke, podatkovne sponke in kontrolne sponke),
- na časovne diagrame - potek signalov, časovno soodvisnost signalov.
- Zanima nas, kakšna je minimalna konfiguracija (najmanjše število potrebnih elementov za izvedbo delojočega sistema),
- kakšna je razvojna programska/materialna oprema (analizatorji, emulatorji, simulatorji, prevajalniki, razvojni sistemi, ...).

3.1 Nekaj splošnih podatkov o mikroprocesorju MC6800

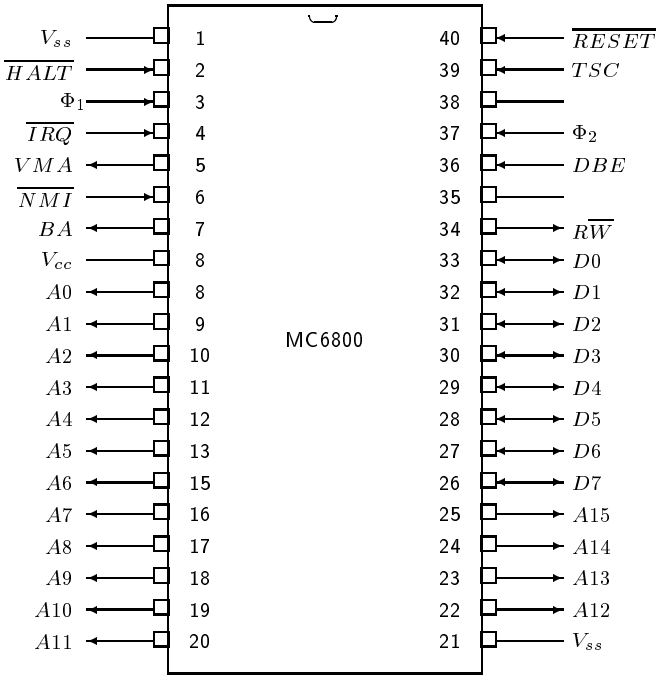
- 6 programsko dostopnih registrov, 7 načinov naslavljanja, 72 ukazov, skupaj z načinom naslavljanja 197 operacijskih kod,
- sklad, vektorski sistem prekinitev,
- 16 (enosmernih) naslovnih sponk za šestnajstbitno naslovno vodilo, 8 obojesmernih podatkovnih sponk za osembitno podatkovno vodilo,
- enojno napajanje +5 Voltov, dvovrstična razporeditev sponk, ohišje s 40 sponkami.

3.2 Opis sponk oziroma signalov

Sponke oziroma signale mikroprocesorja označujemo s prečno črto ali brez prečne črte nad črkovnim imenom. Prečna črta nad imenom pomeni, da je aktivno stanje signala nizko (tudi Low ali L), kar predstavlja logično stanje 0. Če nad imenom ni prečne črte, je aktivno stanje visoko (High ali H), kar predstavlja logično stanje 1.

Nekatere sponke lahko zavzamejo poleg nizkega ali visokega stanja še tretje stanje - to je stanje visoke impedance (High Z ali HiZ). Sponkam/signalom s to lastnostjo pravimo tristanjske (Tri-State ali TS) sponke/signali.

Puščica označuje smer signala: od ali k mikroprocesorju (izhod ali vhod) ali v obe smeri.



- Φ_1, Φ_2 (Vhoda): Dvofazna ura z neprekrivajočimi se prehodi. Visoko stanje obeh signalov se časovno ne prekriva. Vhoda nista TTL združljiva. Signala smeta zaradi dinamičnih notranjih registrov ostati v nespremenjenem stanju največ $5\mu s$. Najvišja dovoljena frekvenca za osnovno izvedbo mikroprocesorja je 1 MHz.

- $A0 - A15$ (Izhodi, TS) (Address): Sponke za 16-bitno naslovno vodilo pomnilnika in vhodno izhodnih vmesnikov.

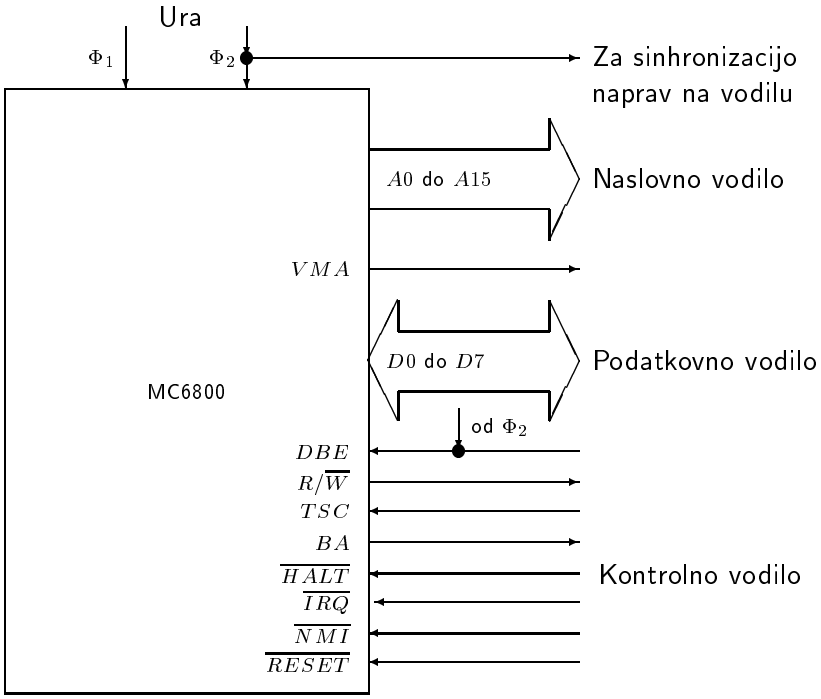
- $D0 - D7$ (Dvosmerni, TS) (Data): Sponke za 8-bitno podatkovno vodilo.

- $\overline{RESET}$ (Vhod): Začetni zagon mikroprocesorja ob prehodu na visok nivo (ali ponovni zagon, če signal vstraja v nizkem stanju vsaj 8 urinih ciklov).

- VMA (Izhod) (Valid Memory Address): Veljaven naslov na naslovnem vodilu. Namreč, v nekaterih strojnih ciklih na naslovnih sponkah ni veljavnega naslova.

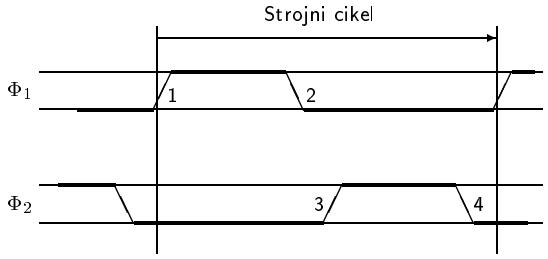
- $\overline{HALT}$ (Vhod): Vse aktivnosti v mikroprocesorju se ustavijo. ($BA \leftarrow H$, $VMA \leftarrow L$, vse sponke TS gredo v tretje stanje - stanje visoke impedance).
- BA (Izhod) (Bus Available): Gre v aktivno (visoko stanje), če se aktivnosti v mikroprocesorju ustavijo, kot posledica $\overline{HALT}$ ali ukaza WAI (WAIt for Interrupt).
- TSC (Vhod) (Tri-State-Control): Signal postavi naslovne sponke in $R/\overline{W}$ v tretje stanje. Sme ostati v aktivnem stanju največ 4.5 μs. Koristi se pri neposrednem dostopu do pomnilnika.
- DBE (Vhod) (Data Bus Enable): Signal postavi podatkovno vodilo (sponke $D0 - D7$) v tretje stanje, če je neaktiven (v nizkem stanju). Uporablja se pri neposrednem dostopu do pomnilnika. Običajno je povezan na urin signal Φ_2.
- $R/\overline{W}$ (Izhod, TS) (Read/Write): Določa smer prenosa po podatkovnem vodilu (Beri/Piši). Beri: k mikroprocesorju, piši: od mikroprocesorja k pomnilniku.
- $\overline{NMI}$ (Vhod) (Non-Maskable-Interrupt): Vhod zahteve za prekinitev, ki se je programsko ne da preprečiti. Vhod je občutljiv na prehod z visokega na nizek nivo.
- $\overline{IRQ}$ (Vhod) (Interrupt Request): Vhod zahteve za prekinitev, ki jo je možno preprečiti z maskirnim bitom v registru CCR ($I = 1$). Občutljiv je na nizek nivo signala in ne na prehod signala.

3.3 Povezava na vodilo



3.4 Urin signal Φ_1, Φ_2

Prehodi obeh signalov ure se časovno ne prekrivajo. Oba signala nista nikoli istočasno v visokem stanju. Potek signala ure sodobnejših mikroprocesov je preprostejši.

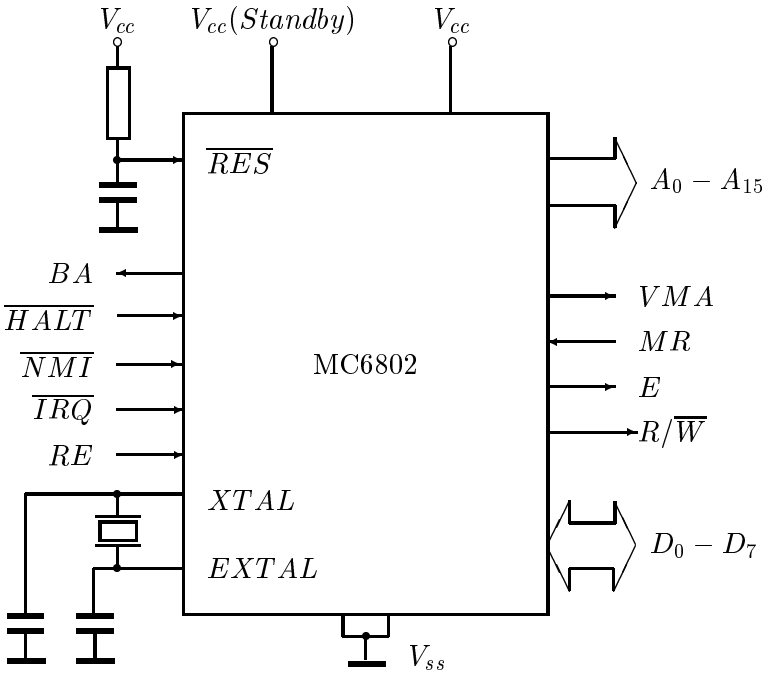


Strojni cikel mikroprocesorja MC6800 je po trajanju enak urinemu ciklu. V enem strojnem ciklu se na primer izvede branje iz pomnilnika, pisanje v pomnilnik ali poljubna notranja operacija mikroprocesorja. V enem strojnem ciklu se realizirajo 4 mikrooperacije. En ukazni cikel obsega dva ali več strojnih ciklov.

- V trenutku **1** gre naslov (običajno vsebina programskega števca) na naslovno vodilo,
- v trenutku **2** se (če gre za ukazno prevzemni cikel) poveča vsebina programskega števca,
- v trenutku **3** gredo podatki na podatkovno vodilo, smer določa signal $R/\overline{W}$,
- v trenutku **4** je podatek prevzet s strani mikroprocesorja ali s strani druge enote (pomnilnika) na vodilu.

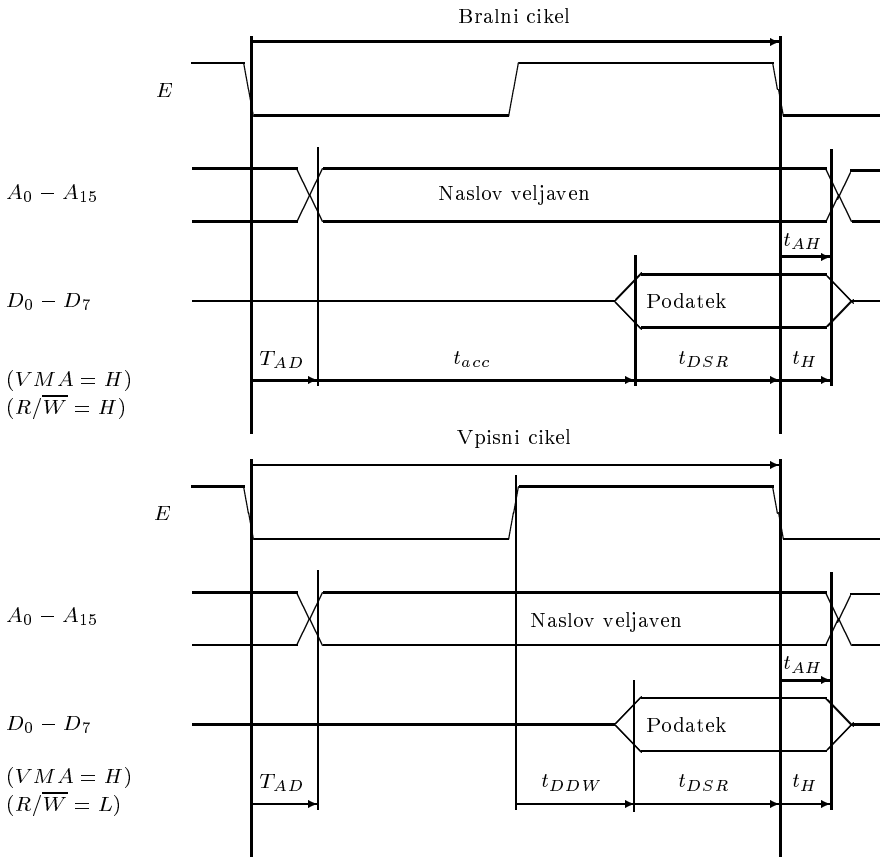
3.5 Mikroprocesor MC6802

Mikroprocesor MC6802 je izpopolnjen naslednik mikroprocesorja MC6800. Njegovi signali so večinoma enaki signalom mikroprocesorja MC6800, vendar nima signalov TSC, DBE, Φ_1, Φ_2, ima pa nekatere nove: $Xtal, EXtal, MR, E(\Phi_2), RE, V_{cc}(StandBy)$. Bistvena razlika mikroprocesorja MC6802 od njegovega predhodnika je pravzaprav v tem, da vsebuje generator urinega signala, da je oblika urinega signala bolj enostavna, in da ima 128 bajtov notranjega pomnilnika RAM (na naslovih \$0000 do \$007F). Obstaja različica MC6808, ki nima notranjega pomnilnika RAM in seveda tistih sponk, ki se nanašajo nanj. S stališča programerja sta oba (MC6802 in MC6808) popolnoma enaka mikroprocesorju MC6800.



Slika prikazuje signale mikroprocesorja MC6802. Pomen večine signalov spoznamo po oznakah; njihov pomen je enak signalom mikroprocesorja MC6800 in tega ne bomo ponavljali. Na sponki $XTAL$ in $EXTAL$ priključimo kristal frekvence $4MHz$ (tipično), ki stabilizira notranji oscilator. Ta niha s štirikrat višjo frekvenco kot signal E. Pravokotni signal E služi za sinhronizacijo enot, ki so priključene na vodilo. Strojni cikel procesorja je enak eni periodi signala E (tipično traja eno mikrosekundo). Toliko traja tudi pomnilniški cikel (beri ali piši). Časovno soodvisnost signalov na vodilu v trajanju enega pomnilniškega cikla prikazuje naslednja slika. Časovni potek teh signalov je važen pri priključitvi pomnilnih vezij in v/i vmesnikov.

Pomnilniški cikel začne, ko zavzame signal E veljaven nizek nivo in traja, dokler se signal ponovno ne vrne z visokega na veljaven nizek nivo. Naslovni signali $A_0 - A_{15}$, signal $R/\overline{W}$ in signal VMA zavzamejo veljavno stanje kasneje, in sicer po času t_{AD} (Address Delay), ki ni nikoli daljši od 270 ns. V bralnem ciklu postavi podatkovne signale pomnilno vezje. Ti signali zapustijo stanje visoke impedance in postanejo veljavni po času dostopa t_{acc}, ki je karakteristika pomnilnega vezja. Čas dostopa do pomnilnika pri trajanju pomnilniškega cikla $t_{cyc} = 1\mu s$ ne sme preseči $t_{acc} = 530$ ns, čemur današnja pomnilna vezja z lahkoto zadostijo. Procesor bere stanje podatkovnih signalov v trenutku, ko signal E zapusti visoko stanje, signali pa morajo biti v veljavnem stanju že pred tem, in sicer najmanj $t_{DSR} = 100$ ns (Data Setup Time Read). Pomnilnik mora držati podatek veljaven vsaj še nekaj časa po odčitku (vsaj 10 ns). Ta držalni čas podatka je na sliki označen s t_H.



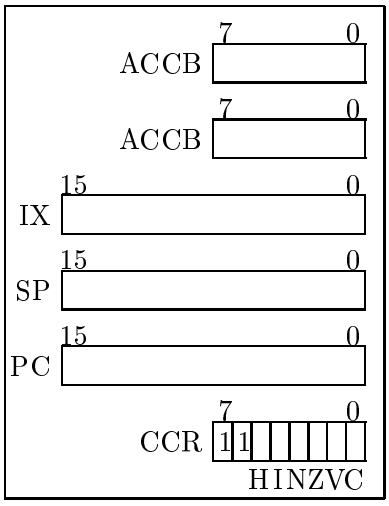
Razmere pri vpisu podatka v pomnilnik so podobne; signal $R/\overline{W} = L$ in smer podatkov je obrnjena od procesorja proti pomnilniku. Procesor postavi podatek na vodilo ob prehodu signala E navzgor, signali $D_0 - D_7$ pa postanejo veljavni najkasneje po času $t_{DDW} = 225\ ns$ (Data Delay Write). Podatek pa se mora vpisati v pomnilnik najkasneje tedaj, ko se signal E vrne v nizko stanje.

Dostop do počasnejših pomnilnih vezij omogoča signal MR (Memory ready) in se običajno za stalno veže na visok nivo. Če je ta signal v nizkem stanju, se pomnilniški cikel podaljša (signal E ostane v visokem stanju), vendar pa ne sme vztrajati v nizkem stanju več kot 4.5 mikrosekunde.

Nov je še signal RE (Ram Enable) in kot pove ime, v visokem stanju omogoči notranji pomnilnik RAM, v nizkem stanju pa ga onemogoči (mikroprocesor nima več dostopa do svojega notranjega pomnilnika). Notranji RAM lahko stalno napajamo (sponka $V_{cc}StandBy$), če želimo, da se njegova vsebina (spodnjih 64 bajtov) z izklopom glavnega napajanja ne izgubi.

3.6 Programski model mikroprocesorja MC6800/MC6802

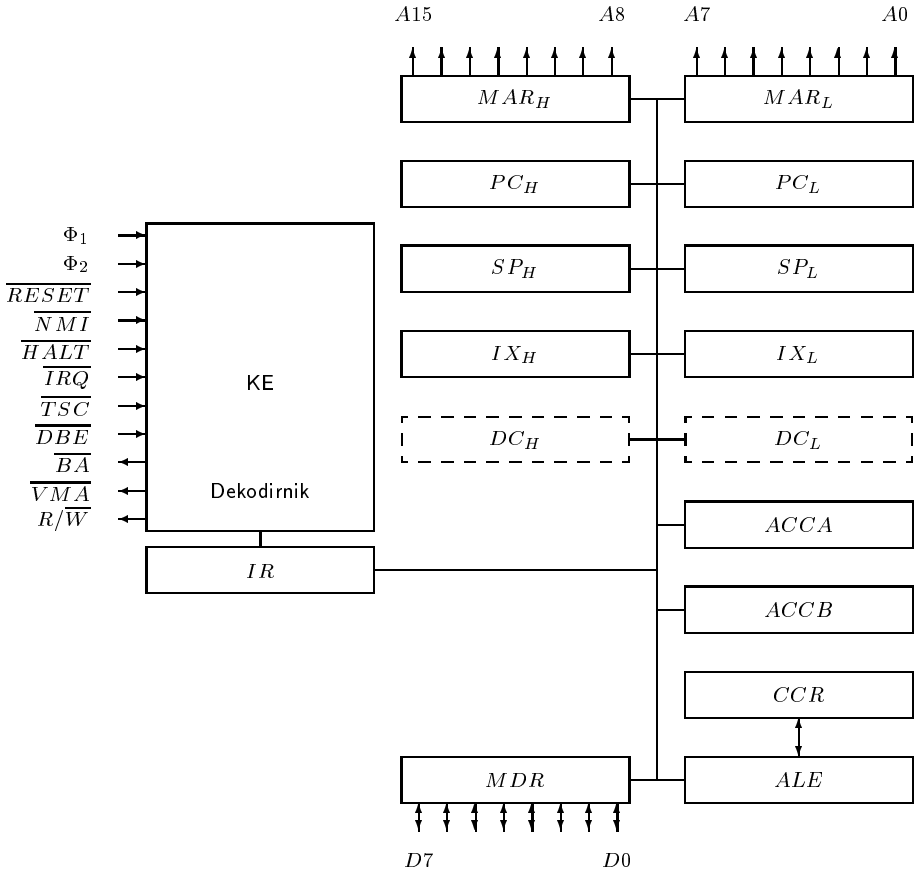
Registri, ki so programsko dostopni, ki jih 'vidi' programer oziroma ima možnost vplivati na njihovo vsebino z ukazi, sestavljajo **programski model** mikroprocesorja.



- **ACCA**: 8-bitni akumulator A (v zbirnem jeziku označen z A),
- **ACCB**: 8-bitni akumulator B (v zbirnem jeziku označen z B),
- **IX**: 16-bitni indeksni register (v zbirnem jeziku označen z X),
- **SP**: 16-bitni skladovni kazalec (v zbirnem jeziku označen s S),
- **PC**: Programski števec (Program Counter) (16 bitov),
- **CCR** 8-bitni register stanj (Condition Code Register) (efektivno 6 bitov),
  - **H** - bit polovičnega prenosa (naprej z bita b_3) (Half Carry),
  - **I** - maskirni bit zahteve za prekinitev na sponki $\overline{IRQ}$ (Interrupt Mask),
  - **N** - bit negativne vrednosti (Negative bit),
  - **Z** - bit ničelne vrednosti (Zero bit),
  - **V** - bit presega območja računaja (-128 do +127) (Overflow),
  - **C** - bit prenosa (naprej z bita b_7) (Carry).

3.7 Organizacija mikroprocesorja MC6800/MC6802

Mikroprocesor MC6800 ima eno samo 8-bitno notranje vodilo, ki povezuje notranje 8-bitne in 16-bitne registre.



Poleg programsko dostopnih registrov ima mikroprocesor:

- **IR**: 8-bitni ukazni register, ki hrani kodo (določilo) operacije, (Instruction Register),
- **DC**: 16-bitni podatkovni števec, ki hrani določilo operanda (Data Counter),
- **MAR**: 16-bitni naslovni (vmesni) register pomnilnika, ki povezuje notranje vodilo z zunanjim naslovnim vodilom,
- **MDR**: 8-bitni obojesmerni podatkovni (vmesni) register pomnilnika, ki povezuje notranje vodilo z zunanjim podatkovnim vodilom.

3.8 Potek izvajanja ukazov

Primer:
Shrani vsebino akumulatorja **A** v pomnilniško besedo z naslovom $203.

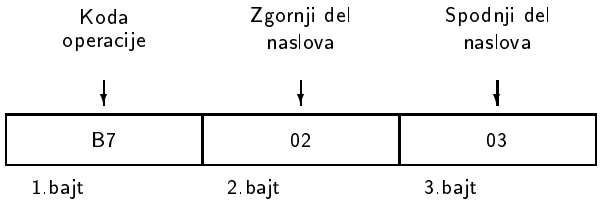
Operacijo bi lahko simbolično zapisali tudi takole:

$$(\$203) \leftarrow A \quad \text{in} \quad PC \leftarrow PC + 3$$

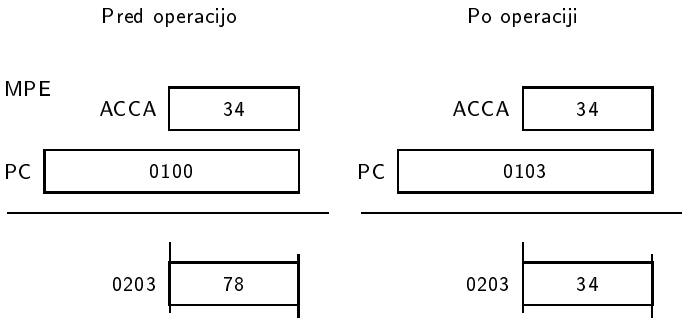
ali s črkovno kodo operacije *shrani vsebino akumulatorja A* - *STAA* (Store Accumulator A),

$$STAA \qquad \$203.$$

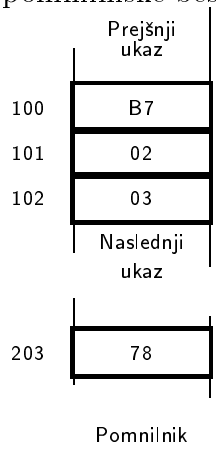
Ukaz obsega tri bajte. Prvi bajt vsebuje določilo operacije skupaj z načinom naslavljanja (Operacijsko kodo), druga dva bajta sta 16-bitni naslov operanda (najprej zgornji del in nato spodnji del naslova).



Naj bo vsebina akumulatorja A enaka $34 in vsebina pomnilniške besede z naslovom $203 naj bo $78. Poglejmo, kako izvršitev obravnavanega ukaza vpliva na registre mikroprocesorja (PC in ACCA) ter na izbrano pomnilniško besedo.



Ukaz naj je shranjen v tri zaporedne pomnilniške besede z naslovi $100, $101in$102.



Zapišimo zaporedje mikrooperacij za izvedbo ukaza.

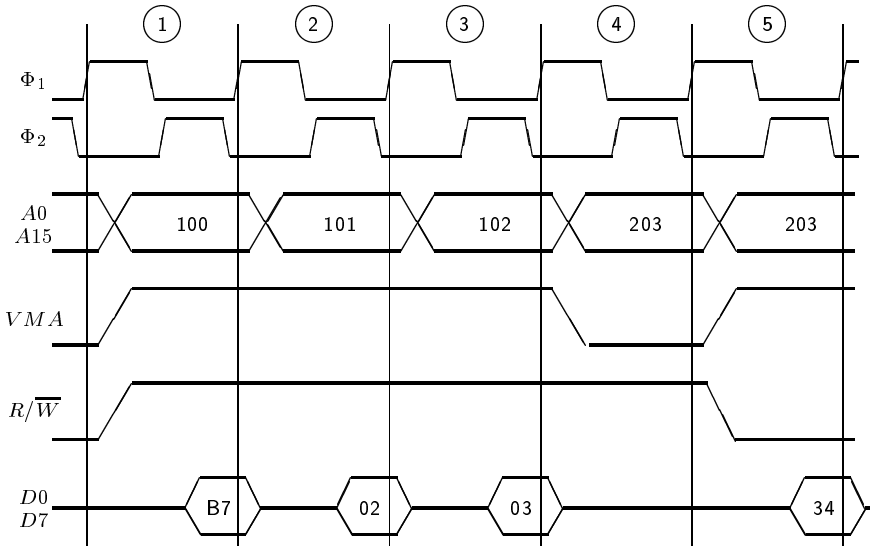
	Začetek izvajanja ukaza, PC = 0100
$\Phi_1 \uparrow$	$MAR \leftarrow PC,\ \ VMA \leftarrow H,\ \ R/W \leftarrow H$
$\Phi_1 \downarrow$	$PC \leftarrow PC + 1$
$\Phi_2 \uparrow$	omogoči podatku pot na vodilo (Φ_2 na vodilo),
$\Phi_2 \downarrow$	$MDR \leftarrow (MAR),\ \ IR \leftarrow MDR$ in dekodiranje
$\Phi_1 \uparrow$	$MAR \leftarrow PC,\ \ VMA \leftarrow H,\ \ R/W \leftarrow H$
$\Phi_1 \downarrow$	$PC \leftarrow PC + 1$
$\Phi_2 \uparrow$	omogoči podatku pot na vodilo,
$\Phi_2 \downarrow$	$MDR \leftarrow (MAR),\ \ DC_H \leftarrow MDR$
$\Phi_1 \uparrow$	$MAR \leftarrow PC,\ \ VMA \leftarrow H,\ \ R/\bar{W} \leftarrow H$
$\Phi_1 \downarrow$	$PC \leftarrow PC + 1$
$\Phi_2 \uparrow$	omogoči podatku pot na vodilo,
$\Phi_2 \downarrow$	$MDR \leftarrow (MAR),\ \ DC_L \leftarrow MDR$
	Ukazno prevzemni cikel je s tem končan
$\Phi_1 \uparrow$	$MAR \leftarrow DC,\ \ VMA \leftarrow L,\ \ R/W \leftarrow H$
$\Phi_1 \downarrow$	PC se ne poveča, to je izvršilni cikel
$\Phi_2 \uparrow$	Φ_2 bi omogočil podatku pot na vodilo, a VMA to prepreči
$\Phi_2 \downarrow$	neveljaven podatek na podatkovnem vodilu
$\Phi_1 \uparrow$	$VMA \leftarrow H,\ \ R/\bar{W} \leftarrow L$
$\Phi_1 \downarrow$	PC se ne poveča, to je izvršilni cikel
$\Phi_2 \uparrow$	$MDR \leftarrow A,\ \ \Phi_2$ omogoči podatku pot na vodilo,
$\Phi_2 \downarrow$	$(MAR) \leftarrow MDR$, vpis v pomnilnik
	Izvršilni cikel in s tem ukazni cikel je končan

Izvajanje ukaza traja pet urinih (in hkrati strojnih) ciklov, tri cikle traja prevzem ukaza iz pomnilnika, dva cikla sta potrebna za izvršitev ukaza. Očitno mikroprocesor ne zmore v enem strojnem ciklu prenesti 16-bitnega naslova (dva bajta) na naslovno vodilo in 8-bitnega podatka iz akumulatorja na podatkovno vodilo po notranjem 8-bitnem vodilu.

3.9 Povzetek izvajanja ukaza

Število strojnih ciklov	Zaporedna številka cikla	Stanje signala VMA	Naslovno vodilo	Stanje signala $R/\bar{W}$	Podatkovno vodilo
5	1	H	0100	H	B7 (Op.Koda)
	2	H	0101	H	02 (Del Naslova)
	3	H	0102	H	03 (Del Naslova)
	4	L	0203	H	-
	5	H	0203	L	34 (Operand)

3.10 Časovni potek signalov na vodilu



4 Zbirni jezik

Računalnik naredi uporaben šele programska oprema. V grobem delimo programsko opremo na sistemsko programsko opremo in na uporabniško programsko opremo. Del sistemske programske opreme je razvojna programska oprema. Med razvojno programsko opremo sodijo tudi prevajalniki oziroma *programski jeziki*.

Računalnik pozna le strojni jezik in vsak program je na koncu zapisan (preveden) v strojni jezik. Program v strojnem jeziku je dejansko zaporedje ničel in enic, ki ga računalnik (procesna enota) obravnava kot zaporedje ukazov in podatkov. Ničesar ni, kar bi vsebino pomnilniške besede opredelilo za ukaz. Vsebino besede procesor tolmači kot ukaz, kadar programski števec vsebuje njen naslov.

Nihče ne programira v strojnem jeziku. Programiranje v strojnem jeziku je *prenaporno*, *neučinkovito*, *nepregledno* in *nerazumljivo*. Naslednji binarni zapis bi lahko predstavljal kos programa (zaporedja ukazov) v strojnem jeziku mikroprocesorja MC6800. Kaj bi pomenil?

```
1 0 0 0 0 1 1 0
1 0 1 0 1 0 1 0
1 0 0 1 0 1 1 1
0 1 0 1 0 1 0 1
```

(Zapis pomeni dva ukaza. Prvi dve vrstici sta za prvi ukaz: napolni akumulator A z vrednostjo AA_{16}. Drugi dve vrstici sta za drugi ukaz: shrani vsebino akumulatorja na naslov 55_{16}. Posamezne vrstice si razlagamo kot vsebine pomnilniških besed.)

Krajši in predvsem preglednejši je zapis strojnega programa v šestnajstiški obliki. Za zgornji primer:

```
86
AA
97
55
```

Šestnajstiški zapis je praktičen pri čiščenju programov (Debugging) na nivoju strojnega jezika. V šestnajstiški obliki bi se dalo (in se da) pisati celo prave programe. Za pretvorbo v strojno obliko in za vnos programa v pomnilnik pa bi rabili ustrezen pripomoček (šestnajstiški nalagalnik).

Zapis v šestnajstiški obliki je sicer lažje čitljiv (bolj pregleden), a še vedno prav tako (ali skoraj tako) težko razumljiv kot v binarni - strojni obliki. Vprašajmo se, kaj sploh pomeni zapis 86? Šestnajstiški zapis 86 se sicer lažje obravnava kot binarni zapis 10000110, a pomena se iz njega (večinoma) ne vidi.

Naslednji korak bližje razumevanju človeka je predstavitev (zapis) operacij, ki jih naredi računalnik, z zapomnljivkami ali **mnemoniki**. Po mnemonikih si človek lažje zapomni operacije, ki (naj) jih naredi računalnik. Mnemonike (črkovne kode operacij) predpiše proizvajalec, zato za isto operacijo v praksi srečamo različne črkovne kode. Recimo za operacijo: *napolni (nek register)* se pojavljajo mnemoniki: LDAA, LDA, LD, MOV, MOVE, i.t.d., kar izhaja iz angleških besed **Load** in **Move**. Nabor ukazov (Operacijskih kod) skupaj z mnemoniki daje proizvajalec med ostalo dokumentacijo MPE. Za naš primer:

```
LDAA #
AA
STAA
55
```

S tabelo mnemonikov se da z malo truda prekodirati ('prevesti') mnemoničen zapis v strojno obliko, to je, da se sestaviti ali 'zbrati skupaj' (ali kot smo se navadili reči: asemblirati) strojni program. Postopek prekodiranja je enolično določen in ga hitreje, predvsem pa brez pomot, opravi računalniški program. Orodju (programu) za zbiranje ali prevajanje rečemo **zbirnik** (Assembler). Zaporedju ukazov, ki je zapisano z mnemoniki ukazov, rečemo zbirniški program ali program v zbirnem jeziku.

Namesto da pišemo en ukaz v več vrstic, s čimer hočemo zgolj ponazoriti dejstvo, da en ukaz obsega več pomnilniških besed, zapišemo cel ukaz v eno vrstico. Pravilo: en ukaz - ena vrstica je tipično za vse zbirne jezike. Pravimo, da so zbirni jeziki vrstično usmerjeni in imajo odnos 1 : 1 s strojnimi jeziki. V našem primeru:

```
LDAA   # $ AA
STAA   $ 55
```

Pred številski konstanti smo zapisali še znak $, s čimer smo poudarili, da gre za števili v šestnajstiški obliki.

Mnemoniki so sestavni del zbirnega jezika. Tudi znak $ je v našem primeru eden od posebnih (črkovnih) znakov zbirnega jezika. Zbirnik in zbirni jezik pa omogočata še veliko več, kot pisanje mnemonikov namesto zaporedij ničel in enic. Na primer, konstante in naslove v ukazih zapišemo simbolično, dejanske vrednosti pa jim priredimo na bolj vidnem (preglednem) delu programa, tipično na začetku programskega besedila, ali pa prirejanje popolnoma prepustimo zbirniku. Kako to dosežemo, bomo spoznali kasneje. Za naš primer:

```
LDAA   # NEKAJ
STAA   NASLOV
```

Programski jezik, v katerem programiramo, imenujemo izvorni (Source) jezik, programu zapisanem v tem jeziku pa izvorni program (Source). Izvornemu programu, ki je preveden v strojno obliko, pravimo ciljni (object) program.

4.1 Zbirni jezik mikroprocesorja MC6800/MC6802

Edini pravi način za učenje programskega jezika je programiranje v tem jeziku. To velja tudi za zbirni jezik. Zbirni jezik je skoraj nemogoče obravnavati ne da bi se prej opredelili za konkreten procesor oz. računalnik. Programiranje v zbirnem jeziku zahteva temeljito poznavanje arhitekture računalnika, ki je od računalnika do računalnika različna. Velja pa tudi obratno, programiranje v zbirnem jeziku nam pomaga pri razumevanju arhitekture računalnika.

Pri programiranju v zbirnem jeziku moramo biti pozorni na sledeče:

- programski model MPE,
- nabor ukazov (prenosne, aritmetične in logične, skoki, vejitve, i.t.d.),
- načine naslavljanja, kakšni so in kako jih zapišemo v zbirnem jeziku,
- sistem prekinitev, kot ga vidi programer, (začetni zagon, zunanje prekinitve, programske prekinitve, ...),
- možnosti glede vhodno izhodnih prenosov podatkov,
- slovnična pravila zbirnega jezika,
- posebne ukaze (Pseudo Instructions ali Directives),
- posebne (črkovne) znake,
- kakšne enostavne aritmetične (logične) operacije omogoča zbirnik,
- ali omogoča makro ukaze,

- ali omogoča ločeno prevajanje posameznih delov programa,
- ali omogoča pogojno prevajanje (enega ali drugega kosa programa), i.i.d.

Zbirni jezik v grobem določajo:

- pravila pisanja programov (sintaktična pravila),
- ukazi (mnemoniki ukazov, ki jih predpiše proizvajalec računalnika oziroma procesne enote),
- posebni ukazi (PseudoInstructions), imenovani tudi psevdoukazi, zbirniške direktive ali navodila, to so navodila zbirniku, kako naj prevede zbirniški program v strojni jezik,
- posebni znaki za pripis baze številskega sistema konstant, za označevanje načinov naslavljanja, za ločevanje posameznih polj v vrstici in podobno,
- nekatere enostavne računske operacije.

4.2 Delitev ukazov

Mikroprocesor MC6800 pozna 72 ukazov. Ker je z isto operacijo možnih več načinov naslavljanja, način naslavljanja pa je pridružen operacijski kodi, ima procesor 197 različnih operacijskih kod. Ker je operacijska koda osembitna, je torej 'prostih' še 256 - 197 = 59 kombinacij.

Motorola deli nabor ukazov mikroprocesorja na:

- akumulatorske in pomnilniške ukaze: aritmetične, logične, prenosne in testne,
- ukaze z indeksnim registrom in s skladovnim kazalcem,
- skočne in vejitvene ukaze,
- ukaze za nadzor nad stanjem MPE, to so ukazi z registrom CCR in
- ukazi za skok/vejitev v podprogram, povratek iz podprograma ali prekinitvenega programa, ukaz za programsko prekinitev in čakanje na prekinitev.

4.2.1 Nabor ukazov mikroprocesorja MC6800/MC6802

nadaljevanje

4.2.2 Oblike ukazov

Ukazi mikroprocesne enote MC6800 obsegajo od enega do treh bajtov - pomnilniških besed. Torej so različnih dolžin. V prvem (ali edinem) bajtu je določilo operacije, način naslavljanja in, kadar je operand dan s samo operacijo (implicitno), tudi določilo operanda.

Izvajanje ukaza traja najmanj dva strojna cikla in največ dvanajst strojnih ciklov.

4.3 Načini naslavljanja (Addressing Modes)

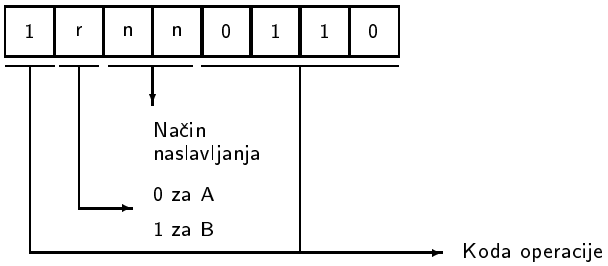
Mikroprocesorska enota MC6800 zmore sedem načinov naslavljanja:

- vsebovano naslavljanje (Inherent, Implied Addressing),
- akumulatorsko naslavljanje (Accumulator Addressing),
- takojšnje naslavljanje (Imidiate Addresing),
- neposredno (direktno) naslavljanje (Direct Addressing),
- (neposredno) razširjeno naslavljanje (Extended Addressing),
- Indeksno naslavljanje (Indexed Addressing) in
- relativno naslavljanje (Relative Addressing).

Vsebovano in akumulatorsko naslavljanje smemo šteti za en sam način naslavljanja. Podobno ni bistvene razlike med neposrednim direktnim in neposrednim razširjenim načinom naslavljanja. Ker ima tudi relativno naslavljanje zelo omejene možnosti, je različnih načinov naslavljanj manj.

4.3.1 Razčlenitev operacijske kode

Operacija *Napolni akumulator* (Load Accumulator) ali s črkovno kodo LDAr, kjer je r = { A, B } , je zakodirana takole:



nn	način naslavljanja, ki je možen s to operacijo
00	takojšnje
01	neposredno (direktno)
10	indeksno
11	(neposredno) razširjeno

4.3.2 Vsebovano naslavljanje

Določilo operanda je vsebovano v kodi operacije oziroma je dano z operacijo samo. Ukaz obsega en sam bajt. Primeri: INX, DES, INCA, i.t.d..

4.3.3 Akumulatorsko naslavljanje

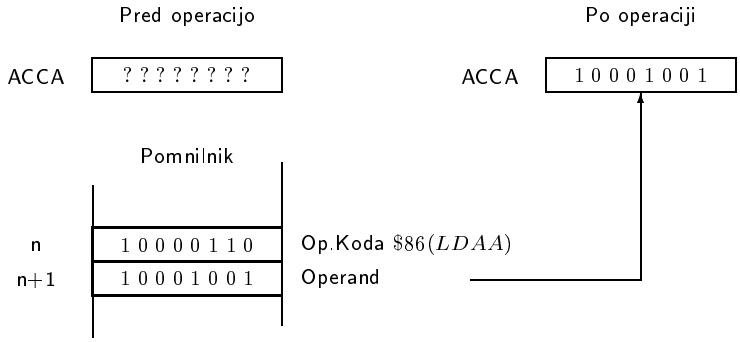
V operaciji sodeluje eden od akumulatorjev (ali oba). Akumulatorsko naslavljanje je pravzaprav poseben primer vsebovanega naslavljanja (recimo CLRA, ABA, PSHB, ...).

4.3.4 Takojšnje naslavljanje

V polju operanda je kar operand sam. Ukaz je dvobajten za enobajten (8-bitni) operand in tribajten za dvobajten (16-bitni) operand.

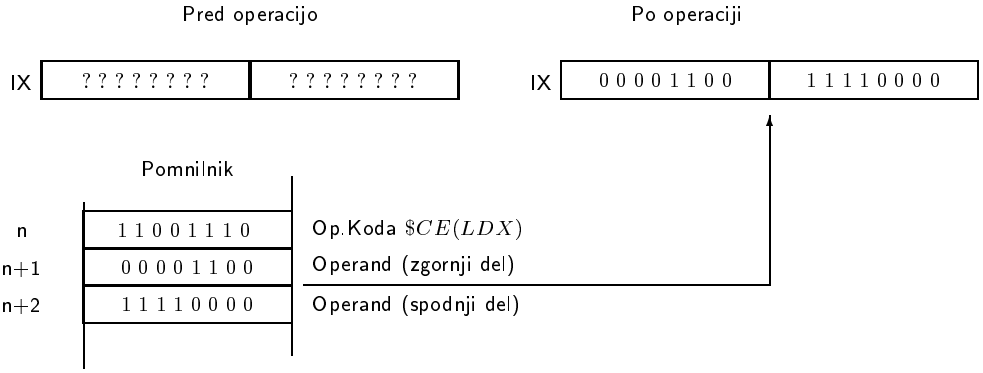
Primer za dvobajten ukaz: napolni akumulator A z vrednostjo 137 (desetiško) ali v zbirnem jeziku

$$LDAA \quad \#137.$$



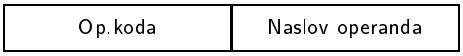
Primer za dvobajten operand: napolni indeksni register z vrednostjo $ 0CF0 ali v zbirnem jeziku

$$LDX \quad \#\$0CF0.$$



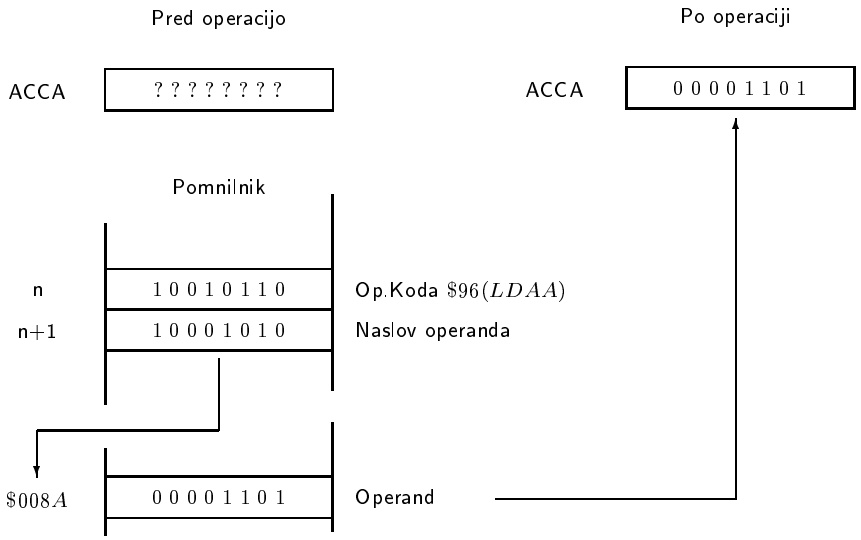
4.3.5 Neposredno (direktno) naslavljanje

V polju operanda je osembitni naslov operanda. Ukaz obsega dva bajta, operand je eno ali dvobajten. Naslovljivo naslovno območje seže v 'prvo stran' pomnilnika (0 do 255).



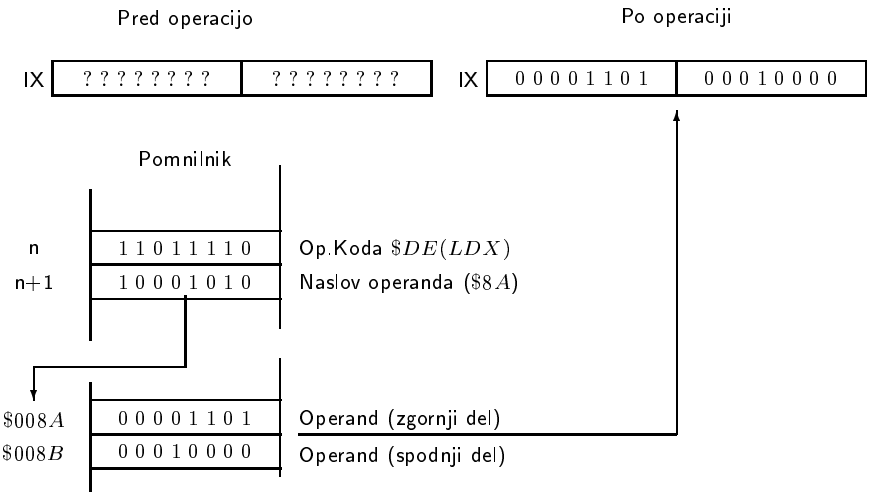
Primer za enobajten operand: napolni akumulator A z vsebino pomnilniške besede z naslovom 138 (desetiško) ali v zbirnem jeziku

$$LDAA \quad 138.$$



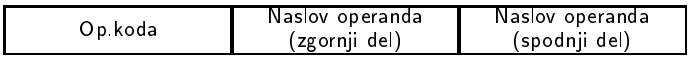
Primer neposrednega naslavljanja za dvobajten operand:
napolni indeksni register z vsebino pomnilniških besed z naslovoma $\$8A$ in $\$8B$
ali v zbirnem jeziku

$$LDX \quad \$8A.$$



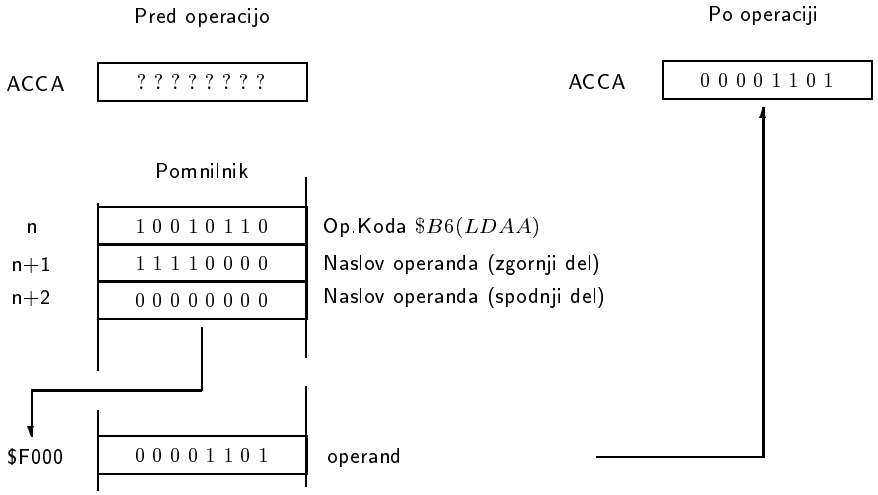
4.3.6 (Neposredno) razširjeno naslavljanje

V polju operanda je dvobajten (16-bitni) naslov operanda. Ukaz obsega tri bajte. Naslovljivo območje je 0 - 65535. Operand je enobajten ali dvobajten.



Primer za enobajtni operand: napolni akumulator A z vsebino pomnilniške besede z naslovom $\$F000$ ali v zbirnem jeziku

$$LDAA \quad \$F000.$$

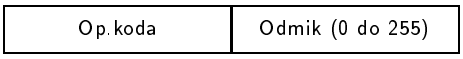


4.3.7 Indeksno naslavljanje

V polju operanda je odmik (Offset), ki skupaj z vsebino indeksnega registra določa dejanski naslov (efektivno adreso) operanda,

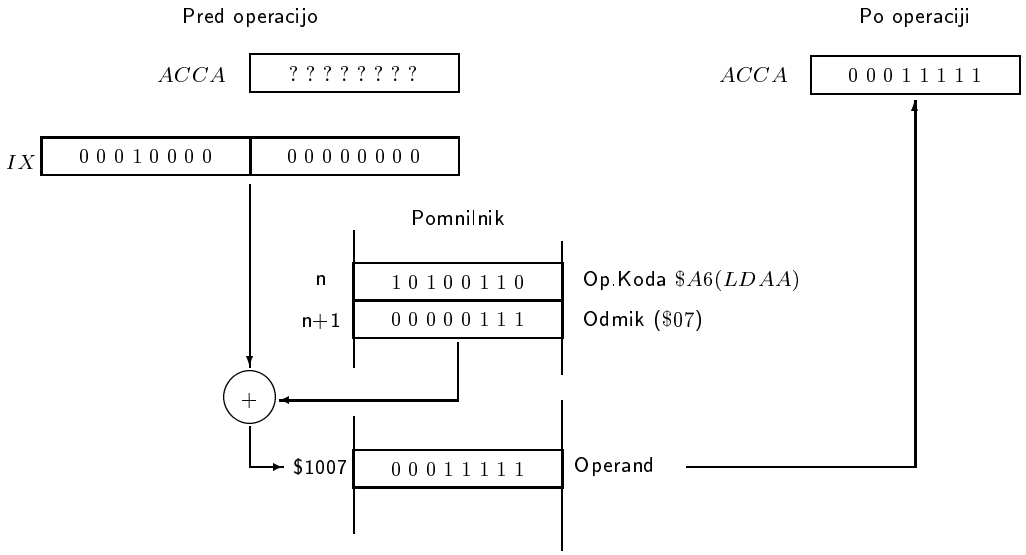
$$EA = IX \quad + \quad odmik.$$

Ukaz je dvobajten. Odmik je 8-bitno število brez predznaka v območju 0 do 255.



Primer: napolni akumulator z vsebino pomnilniške besede z naslovom IX + 7 (odmik) ali v zbirnem jeziku

$$LDAA \quad 7, X.$$

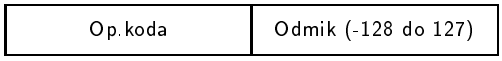


4.3.8 Relativno naslavljanje

V polju operanda je odmik, ki skupaj s trenutno vsebino programskega števca določa dejanski naslov 'operanda',

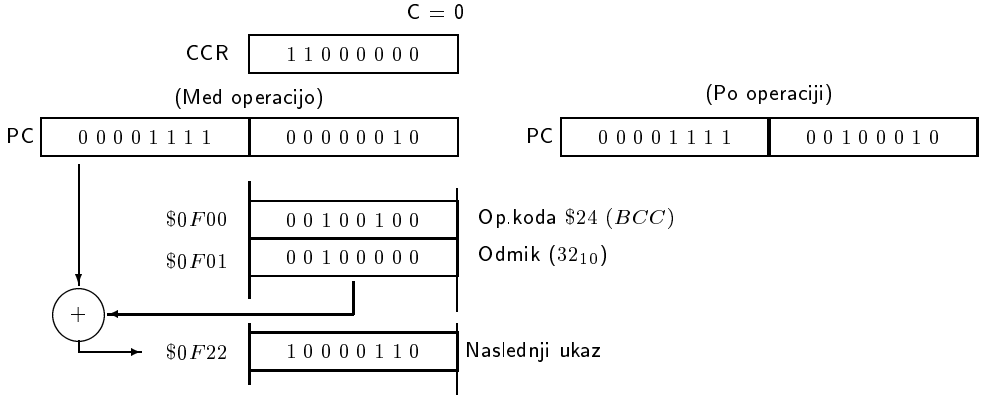
$$EA \;=\; PC \;+\; odmik.$$

Ukaz je dvobajten. Odmik je 8-bitno število v dvojiškem komplementu (-128 do +127). Ta način naslavljanja je možen samo pri vejitvenih (Branch) ukazih.



Primer: če je vrednost bita C v registru stanj enaka 0, nadaljuj izvajanje ukaza na naslovu PC + 32 (Odmik je 32), sicer z naslednjim ukazom ali v zbirnem jeziku

$$BCC \quad 32$$



Ker po prevzemu tekočega ukaza iz pomnilnika programski števec že vsebuje naslov naslednjega ukaza, ki neposredno sledi tekočmu ukazu, je območje 'skokov'

(n + 2 - 128) do (n + 2 + 127),

kjer je n prvi naslov dvobajtnega vejitvenega ukaza. Torej je območje skokov od -126 do + 129 relativno na naslov vejitvenenega ukaza.

4.3.9 Pregled načinov naslavljanja

Način naslavljanja	Dolžina ukaza v bajtih	Dolžina naslova operanda v bajtih	Dolžina operanda v bajtih
Vsebovano	1	-	1 ali 2
Neposredno (direktno)	2	1 (0 do 255)	1 ali 2
(Neposredno) razširjeno	3	2 (0 do 65535)	1 ali 2
Indeksno	2	2 (odmik 0 do 255)	1 ali 2
Takojšnje	2 ali 3	2 (naslov Op.kode+1)	1 ali 2
Relativno	2	2 (odmik -126 do +129)	naslov ukaza

4.4 Programiranje v zbirnem jeziku za MC6800/MC6802

Osnovna enota programa v zbirnem jeziku je vrstica (ali stavek). Zbirni jezik je vrstično usmerjen. Vrstica je naprej deljena na polja:

OZNAKA	**OPERATOR**	**OPERAND**	**;POJASNILO**
Na primer:			
ZANKA	LDAA	#10	;vnesi v akumulator A vrednost 10

Zaporedje posameznih polj znotraj vrstice je važno. Polja ločujejo ločilni znaki. V našem primeru je ločilni znak presledek ali več zaporednih presledkov. Skoraj vsako polje sme biti prazno - razen polje OPERATORja.

Polje **OZNAKA** (Label) vsebuje simbol ali pa je prazno. Simbol (simbolično ime) je poljubno zaporedje črk in številk najdaljše dolžine 6, začenši **obvezno** s črko in brez praznega prostora (presledkov) v začetku vrstice. V našem primeru je ime simbola **ZANKA**. To je simbolična oznaka za (začetni) **naslov** ukaza **LDAA**.

Priporočljivo je izbirati taka simbolična imena, ki sama sebe pojasnjujejo, na primer START, če gre za prvi ukaz v programu. Stavek, ki nima oznake, **mora** imeti pred **OPERATOR**jem prazen prostor (eden ali več presledkov). Skratka, polje oznake je v tem primeru prazno. V programu je največ takšnih vrstic. Tipični primeri uporabe simboličnih oznak vrstic (označevanja vrstic) so programske zanke, pogojni skoki (razvejitve programa), poimenovanje podprogramov (subrutin) in označevanje naslovov podatkov.

Polje **OPERATOR** vsebuje mnemonik ukaza (operacijske kode) mikroprocesorja ali pa posebni ukaz (*Pseudo Instruction*). Mnemonik ukaza določa operacijo, ki jo zna (in mora) opraviti mikroprocesor med izvajanjem programa. Te predpiše proizvajalec in jih najdemo v tabelah med dokumentacijo.

Posebni ukaz je navodilo zbirniku (torej prevajalnemu programu), kako naj program iz zbirnega jezika prevede v strojni jezik. Mnemoniki ukazov **se** prevedejo v svoje strojne ekvivalente. Posebni ukazi **se ne** prevedejo - nimajo strojnih ekvivalentov. Torej jih v prevedenem programu ni. Seveda pa bistveno vplivajo na izgled prevedenega programa. Če bi program, ki ga napišemo v zbirnem jeziku, prevajali mi sami, teh ukazov v bistvu ne bi potrebovali.

(Tipični) posebni ukazi našega zbirnika so:

```
        NAM   ime     ;poimenuje program (je obvezen na začetku programa)
        ORG   nnnn    ;postavi števec lokacij na vrednost nnnn
POD     EQU   nnnn    ;priredi simbolu POD vrednost nnnn
        RMB   nnnn    ;predvidi nnnn prostih pomnilniških besed
        FCB   nn      ;predvidi pomnilniško besedo z vsebino nn
        FDB   nnnn    ;predvidi dve zaporedni pomnilniški besedi z vsebino nnnn.
        FCC   'txt'   ;predvidi zaporedje pomnilniških besed z ASCII
                      ;kodiranim črkovnim nizom txt.
        END           ;zaključi programsko besedilo (je obvezen na koncu datoteke).
```

Pojem števca lokacij in način uporabe posebnih ukazov bomo razložili kasneje.

V polju **OPERAND** je bodisi parameter posebnega ukaza bodisi določilo operanda: njegova dejanska vrednost ali simbolično ime. To polje je v ukazih brez določila operanda (enobajtni ukazi) kajpak prazno.

Vrstica konča s pojasnilom. Pojasnilo ni obvezno, je pa zaželeno, kajti program v zbirnem jeziku brez pojasnil je (drugi osebi, pa kdaj kasneje tudi nam samim) mnogo težje razumeti. Pojasila naj bodo kratka in informativna. Pojasnilo: *napolni akumulator A z vrednostjo 10* k ukazu

$$LDAA \quad \#10$$

nima smisla. Pojasilo: *nastavi začetno vrednost števca* k temu istemu ukazu pa je dopolnilno pojasilo in je še kako želeno (daje vrstici pomen).

Naš zbirni jezik nima veliko posebnih znakov. Njihov pomen je razviden iz naslednje preglednice:

Posebni znaki:

$*$	;pojasnilo (komentar) ali števec lokacij
#	;takojšne naslavljanje, npr. ldaa # 11
$	;šestnajstiška vrednost, npr. ldaa # $ 11
%	;dvojiška vrednost, npr. ldaa # % 10101010
	;Zbirnik sicer privzame desetiško vrednost.

Zvezdica $*$ na začetku vrstice (prvi stolpec !) je potrebna za vrstico s pojasnilom (komentarjem). Prazne vrstice v programu (žal in izjemoma) niso dovoljene. Če želite prazno vrstico, uporabite zvezdico v začetku prazne vrstice - brez komentarja. Zvezdica v polju **OPERAND**a pomeni trenutno vrednost števca lokacij.

4.4.1 Načini naslavljanja in zbirni jezik

Znak # pred operandom pomeni, da gre za takojšnje naslavljanje. Primeri:

```
LDAA   # $ 10
LDX    # 0
LDS    # SKLAD
```

Če pred določilom operanda ne stoji ničesar, zbirnik privzame neposreden (če se da direkten, sicer pa razširjen) način naslavljanja. Primeri:

```
LDAA   $ 10
LDX    0
LDS    SKLAD
```

Za indeksno naslavljanje navedemo v polju operanda ime indeksnega registra (X) in pred njim po potrebi odmik. Recimo:

```
LDAA   X
LDX    1,X
STAB   ODMIK,X
```

4.4.2 Nekaj enostavnih zapisov v zbirnem jeziku

Primer 1

Seštej vsebini pomnilniških besed z naslovoma $40 in $41, shrani rezultat na naslov $42.

```
        LDAA    $40
        ADDA    $41
        STAA    $42
```

ali z indeksnim naslavljanjem:

```
        LDX     #$40
        LDAA    0,X
        ADDA    1,X
        STAA    2,X
```

Primer 2

Denimo, da seštevamo nepredznačeni števili in nas zanima, če smo prekoračili obseg računanja (255_{10}). Če smo obseg prekoračili, naj bo rezultat enak 0:

```
        LDAA    $40
        ADDA    $41
        BCC     VREDU   ; napako pomeni bit C = 1
        CLRA            ; napaka
VREDU   STAA    $42     ; v redu
```

Primer 3

Denimo, da seštejemo števili s predznakom in nas zanima, če smo prekoračili obseg računanja (-128 do +127). Če smo, naj bo rezultat enak 0:

```
        LDAA    $40
        ADDA    $41
        BVC     VREDU   ; napako pomeni bit V = 1
        CLRA            ; napaka
VREDU   STAA    $42     ; v redu
```

Primer 4

Seštejmo dve šestnajstbitni števili. Naj bo prvo število na naslovih $80 in $81, drugo število na naslovih $82 in $83, rezultat naj se shrani na naslova $84 in $85. Držimo se dogovora, da je pomembnejši del števila na nižjem naslovu. Denimo, da mora biti naslov prvega ukaza enak $0100. Zahtevani naslov prvega ukaza dosežemo z ukazom zbirniku

$$ORG \quad \$100$$

Ko zbirnik preveja zbirniški program v strojno obliko, to upošteva in (v času prevajanja) postavi 'števec lokacij' na zahtevano vrednost ($100). Zaporedje ukazov prevede tako, da bo namenjeno za namestitev v pomnilnik od naslova $0100 naprej.

Prvi naslov zaporednih pomnilniških besed bomo označili simbolično z ZN. Ena od možnosti, da to storimo, je taka:

```
ZN      EQU   $80     ; simbolu ZN priredimo vrednost $80
        ORG   $100    ; naj bo naslednji ukaz na naslovu $100
        LDAA  ZN+1    ; spodnji del števila
        LDAB  ZN      ; zgornji del števila
        ADDA  ZN+3    ; spodnji del drugega števila
        ADCB  ZN+2    ; upoštevamo morebitni prenos
        STAA  ZN+5    ; shranimo rezultat
        STAB  ZN+4
```

Bolj prav bi ravnali, če bi za podatke, ki zahtevajo prostor v pomnilniku, od zbirnika zahtevali, naj zagotovi pomnilniški prostor za podatke. To dosežemo takole:

```
        ORG   $80     ; prvi naslov zaporedja podatkov
ZN      RMB   6       ; rabimo 6 pomnilniških besed
        ORG   $100    ; naj bo naslednji ukaz na naslovu $100
        LDAA  ZN+1    ; spodnji del prvega števila
        LDAB  ZN      ; zgornji del prvega števila
        ADDA  ZN+3    ; spodnji del drugega števila
        ADCB  ZN+2    ; upoštevamo morebitni prenos
        STAA  ZN+5    ; shranimo rezultat, spodnji del
        STAB  ZN+4    ; in zgornji del
```

4.5 Primer programa

Naslednji zapis je primer celotnega programa v zbirnem jeziku, ki ga je možno prevesti z zbirnikom prevesti v strojno obliko.

```
        NAM   PRIMER
NPOD    EQU   10          ; število podatkov
*
*                         ; tu začne zaporedje ukazov
*
        ORG   $100        ; naslov prvega ukaza naj bo $100
        LDX   #ZNPOD      ; kazalec za naslavljanje podakov
        LDAB  #NPOD       ; števec za štetje podatkov
        CLRA              ; za izračun vsote podatkov
ZANKA   ADDA  0,X         ; prištejemo vrednost podatka
        INX               ; naslov naslednjega podatka
        DECB              ; smo sešteli vse podatke?
        BNE   ZANKA       ; ne - prištej naslednjega
        STAA  0,X         ; da - shrani vsoto tik za tabelo podatkov
TU      JMP   TU          ; neskončna zanka
*
*                         ; Podatkovni del
*
ZNPOD   RMB   NPOD+1      ; prostor za podatke in rezultat
        END
```

Prva vrstica enostavno poimenuje programsko besedilo. Razen tega, da mora biti prva vrstica vsakega programskega besedila in je s tega stališča obvezna, nima važnejšega pomena.

V drugi vrstici smo konstanti z vrednostjo 10 priredili simbolično ime $NPOD$. Tak ali podoben priredilni stavek sme biti kjerkoli v programskem besedilu, vendar naj bo na vidnem mestu; to pa je na začetku programskega besedila. Namesto, da pišemo dejansko vrednost konstante znotraj programskega besedila, lahko namesto nje, povsod kjer je treba, pišemo simbol $NPOD$. Dejansko vrednost bo nadomestil zbirnik med prevajanjem. Torej, če bi kdaj kasneje želeli spremeniti vrednost konstante, bi to storili na enem samem mestu - v priredilnem stavku, ponovno prevedli program in stvar bi bila opravljena.

Naslednje tri vrstice so za pojasnilo, zato začnejo z zvezdico v prvem stolpcu. Pojasnila pišemo zaradi nas samih ali drugih koristnikov našega programa, zbirnik pa take vrstice med prevajanjem enostavno prezre. Prezre pa tudi taka pojasila, ki so morda v zadnjem polju vrstice.

Sledi ukaz $ORG \quad \$100$, ki zahteva, naj zbirnik prevede program tako, da ga bo možno vložiti v pomnilnik od naslova \$100 naprej. Programom, ki jim določimo položaj izvajanja v času prevajanja, pravimo tudi *absolutni* programi.

Naslednje vrstice besedila sodijo k postopkovnemu delu programa. To so ukazi, za katere najdemo mnemonike v tabelah ukazov mikroprocesorja. Med prevajanjem zbirnik mnemonike nadomesti z ekvivalenti strojnega jezika, simbolična imena pa nadomesti z njihovimi dejanskimi vrednostmi, ki jih določi sam, kot to zahteva programsko besedilo.

Izvajanje našega programa začeti z ukazom, ki je na naslovu \$100. Torej, če v programski števec vsilimo vrednost \$100, steče program do konca, ukaz za ukazom. Zaporedju ukazov, ki jih mikroprocesor izvrši med izvajanjem ukaza, pravimo sled programa, spremljanju izvajanja programa ukaz za ukazom pa sledenje programa.

Kaj se dogaja med izvajanjem programa? V našem programu se v indeksni register najprej naloži vrednost simbola $ZNPOD$, ki pomeni naslov prvega podatka v zaporedju $NPOD$ podatkov. V programu smo določili, naj se zaporedje podatkov nahaja v pomnilniku takoj za zaporedjem ukazov. Akumulator B služi za štetje podatkov, ki se seštevajo v akumulatorju A. Po vsakem prištevanju se poveča indeksni register za ena (tako vsebuje naslov naslednjega podatka) in zmanjša akumulator B za ena. Ko doseže akumulator B vrednost nič, pogoj za vejitev na ukaz s simbolično oznako $ZANKA$ ni več izpolnjen in izvajanje programa se nadaljuje z ukazom $STAA \quad 0, X$. Ta ukaz shrani vsebino akumulatorja A (vsoto) v pomnilniško besedo za podatki. Zadnji ukaz je brezpogojni skok na samega sebe. Izvajanje programa se nadaljuje s prav tem ukazom - brezpogojno. Program se torej 'ujame' v brezizhodni zanki.

Zadnja vrstica s posebnim ukazom END je spet obvezna. Ukaz END pove zbirniku, da je tam konec programskega besedila - konec dela za zbirnik.

4.5.1 'Ročno' prevajanje programa (v šestnajstiški zapis)

Zbirnik (običajno) prevaja v 'dveh korakih' oziroma gre v celoti dvakrat skozi programsko besedilo izvornega programa. V prvem koraku zgradi tabelo simbolov (Symbol Table), torej za vsak simbol najde (določi) njegovo dejansko vrednost. V drugem koraku dokonča delo. Mi se pri prevajanju zaradi lažje razumljivosti ne bomo dosledno držali tega zaporedja.

Med prevajanjem si zbirnik pomaga s *števcem lokacij*, ki se ga da do neke mere pojasniti s programskim števcem, vendar ga z njim **ne smemo** enačiti. Navsezadnje je programski števec register procesne enote, števec lokacij pa ena od spremenljivk zbirnika - programa za prevajanje. Eksplicitno vplivamo na vrednost števca lokacij s posebnim ukazom ORG. Večina posebnih ukazov pa implicitno vpliva na spreminjanje števca lokacij (recimo RMB, FCB, ukaz EQU pa ne). Med prevajanjem zaporedja ukazov se (do neke mere) števec lokacij spreminja tako kot programski števec med izvajanjem programa (če le ni skokov). Torej vsebuje naslov ukaza, ki se prevaja.

Prevedimo program iz prejšnjega primera 'ročno' z uporabo tabele ukazov in z upoštevanjem pravil prevajanja. Med prevajanjem bomo v mislih gradili tabelo simbolov. V tabelo simbolov bomo vpisovali imena simbolov in za vsak simbol v programskem besedilu zraven pripisali njegovo dejansko vrednost. V primeru, da take vrednosti ne najdemo oziroma se je ne da določiti, programa ne moremo popolnoma prevesti v strojno obliko. Nedefiniran simbol pomeni torej slovnično napako v programu, ki jo moramo popraviti in znova prevajati. Napaka bi bila tudi, če bi istemu simbolu priredili več kot eno vrednost. Taki napaki rečemo dvojna definicija simbola.

Prvi simbol, ki ga vpišemo v tabelo simbolov, je simbol z imenom $NPOD$. Temu je z ukazom EQU prirejena vrednost 10. Nadaljujemo. Ukaz

$$ORG \quad \$100$$

zahteva, da števec lokacij postavimo na vrednost \$100. Sledi ukaz

$$LDX \quad \#ZNPOD.$$

Trenutna vsebina števca lokacij služi za naslov ukaza, ki je v tem primeru \$100. Simbol $ZNPOD$ vpišemo v tabelo simbolov, vendar njegove vrednosti še ne poznamo. Ukaz obsega tri baje, zato števec lokacij povečamo za 3 (tako bi se povečal tudi programski števec med izvajanjem programa). Naslednji ukaz

$$LDAB \quad \#NPOD$$

prevedemo tako, da je njegov naslov enak \$103. Vrednost simbola $NPOD$ že poznamo. Ko iz tabele ukazov odberemo šestnajstiški ekvivalent operacijske kode mnemonika $LDAB$, je ta dvobajtni ukaz preveden do konca. Imamo:

Naslov	Op.Koda	Operand
0103	C6	0A

Števec lokacij ima sedaj že vrednost \$105. To je naslov ukaza CLRA, ki obsega en sam bajt. Zato števec lokacij povečamo samo za ena. Sedaj ima števec lokacij vrednost \$106, kar je naslov ukaza v naslednji vrstici:

```
ZANKA   ADDA   0,X   ; prištejemo vrednost podatka
```

V ukazu gre za indeksno naslavljanje. Operacijska koda mnemonika je zato AB, odmik pa je nič. V tej vrstici naletimo na simbol $ZANKA$, ki označuje naslov ukaza $ADDA \quad 0, X$. Ime simbola $ZANKA$ vpišemo v tabelo simbolov in mu priredimo vrednost števca lokacij, ki vsebuje naslov prevajanega ukaza, ta pa je \$106. Vrednost simbola $ZANKA$ je tako določena v času prevajanja programa. Ko smo pisali program, nam ni bilo treba misliti na to, kakšen je dejanski naslov ukaza.

Prevajanje nadaljujemo na enak način kot doslej. Ko pridemo do ukaza:

```
        BNE   ZANKA   ; ne - prištej naslednjega
```

ima števec lokacij vrednost \$10A. V vejitvenem ukazu imamo relativen način naslavljanja. Rekli smo, da je odmik določen s pogojem:

$$EA = n + 2 + Odmik$$

EA je v našem primeru vrednost simbola ZANKA in n je naslov ukaza samega. Torej je odmik:

$$Odmik = ZANKA - (\text{števec lokacij} + 2) = \$106 - (\$10A + 2) = -6$$

ali v dvojiškem komplementu \$FA. Vejitveni ukaz v prevedeni obliki pa je:

Naslov	Op.Koda	Operand
010A	26	FA

Prevesti moramo samo še tri vrstice. Naslednji ukaz za vejitvenim obsega dva bajta in števec lokacij povečamo za dva. Naslov skočnega ukaza in vrednost simbola TU, ki ga vpišemo v tabelo simbolov, je \$10E. Skočni ukaz zavzame tri bajte. Ko naletimo na simbol $ZNPOD$ je vrednost števca lokacij \$10E + 3 = \$0111. Simbolu $ZNPOD$ priredimo vrednost \$0111 in ju vpišemo v tabelo simbolov. V predzadnji vrstici

```
ZNPOD   RMB   NPOD+1   ; prostor za podatke in rezultat
```

se nahaja ukaz RMB, ki zahteva $NPOD + 1$ pomnilniških besed in števec lokacij se poveča še za toliko, to je \$0B. S tem je prevajanje končano. Prevedeni program v celoti je tak:

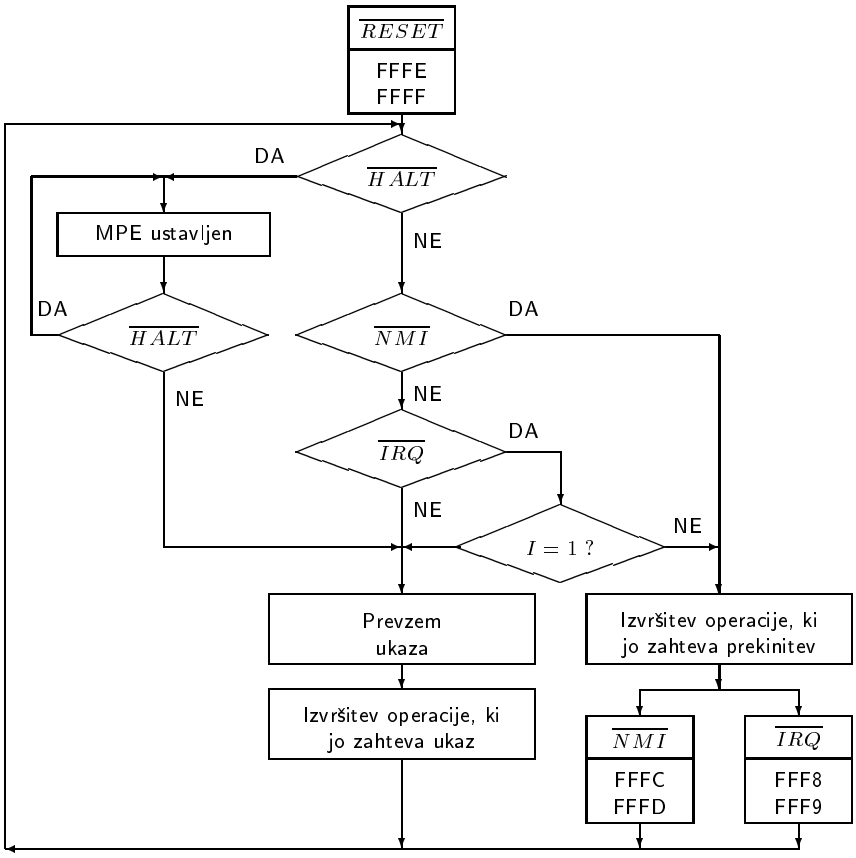
Naslov	Operacijska koda	Operand (zgornji del)	Operand (Spodnji del)
0100	CE	01	11
0103	C6	0A	
0105	4F		
0106	AB	00	
0108	08		
0109	5A		
010A	26	FA	
010C	A7	00	
010E	7E	01	0E
0111			

Tabela simbolov pa je:

NPOD	0A
ZNPOD	01 11
ZANKA	01 06
TU	01 0E

5 Sklad, podprogrami in prekinitve

5.1 Delovanje MC6800/MC6802 - diagram poteka



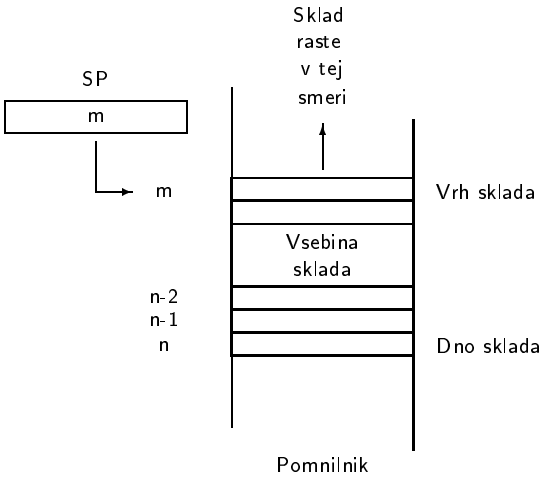
Mikroprocesor izvaja ukaz za ukazom od začetnega zagona do konca delovanja. Ob zunanji zahtevi za prekinitev ($\overline{RESET}, \overline{NMI}, \overline{IRQ}$), se v programski števec naloži vsebina označenih pomnilniških besed, nato pa se začne običajen ukazno prevzemni cikel. Procesor streže zahtevam za prekinitev pred začetkom izvajanja naslednjega ukaza. Ukazi so časovno nedeljivi. Ko se ukaz enkrat začne, se izvede do konca - brez prekinitve.

5.2 Vloga sklada pri mikroprocesorju MC6800/MC6802

V mikroprocesorskem sistemu 6800 je za sklad predviden del glavnega pomnilnika. Tako je pri večini današnjih mikroračunalnikov. Običajno določimo položaj sklada takoj po začetnem zagonu računalnika (ali programa), kasneje pa zelo redko. Vrh sklada (tja kamor vstavljamo in od kjer jemljemo podatke) določa vsebina skladovnega kazalca (SP). Začetni vrh (in hkrati dno) sklada določimo z vpisom primernega pomnilniškega naslova v skladovni kazalec,

LDS #naslov prve pomnilniške besede za sklad - 'dno' sklada

in sklad je sedaj 'prazen'. Z vstavljanjem podatkov v sklad se sklad polni, vrh sklada se pomika proti **nižjim** naslovom. Z jemanjem iz sklada se sklad prazni.



Pomni: skladovni kazalec vedno vsebuje naslov pomnilniške besede, kamor **bo** vstavljen naslednji podatek. Operaciji VSTAVI in VZEMI (denimo PSHA in PULA) sta definirani takole:

PSHA	PULA
$(SP) \leftarrow A$	$SP \leftarrow SP + 1$
$SP \leftarrow SP - 1$	$A \leftarrow (SP)$

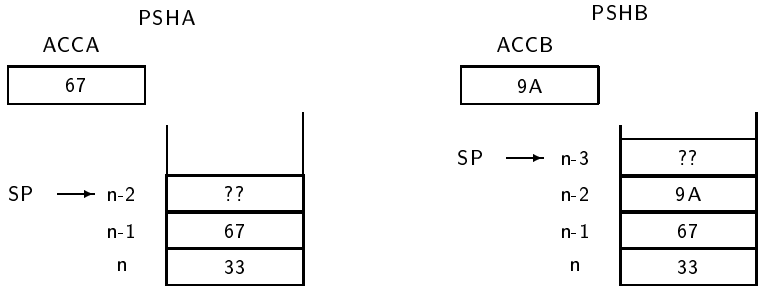
Sklad služi za:

- povezovanje programov s podprogrami: klicni ukazi podprograma za 'skok' v podprogram JSR, 'vejitev' v podprogram BSR in povratek iz podprograma RTS. Sklad omogoča vgnezdenje podprogramov (klic podprograma v podprogramu).
- Streženje zunanjim zahtevam za prekinitev na sponkah $\overline{NMI}$ in $\overline{IRQ}$, streženje programskim zahtevam za prekinitev (ukaz $\overline{SWI}$ - SoftWare Interrupt) in vračanje iz strežnega programa (RTI - ReTurn from Interrupt).
- Shranjevanje (predvsem začasnih) podatkov ($PSHA, PSHB, PULA, PULB$) in za prenos parametrov med klicnim in klicanim programom (podprogramom).
- Daje možnost za načrtovanje položajno neodvisnih (PIC - Position Independent Code) in ponovljivih (Reentrant) programov, torej tudi rekurzivni klic podprograma.

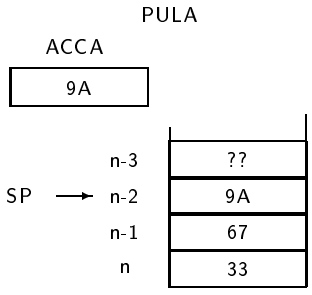
5.3 Shranjevanje podatkov v sklad

V sklad se da (eksplicitno z ukazom) vstaviti ali vzeti samo vsebini obeh akumulatorjev (A in B), ne pa indeksnega registra.

Primer: vstavi v sklad najprej vsebino akumulatorja A (PSHA) in nato še vsebino akumulatorja B (PSHB). Privzemimo, da je vrednost 33 v skladu že od prej.



Primer: vzemi iz sklada podatek in ga shrani v akumulator A.



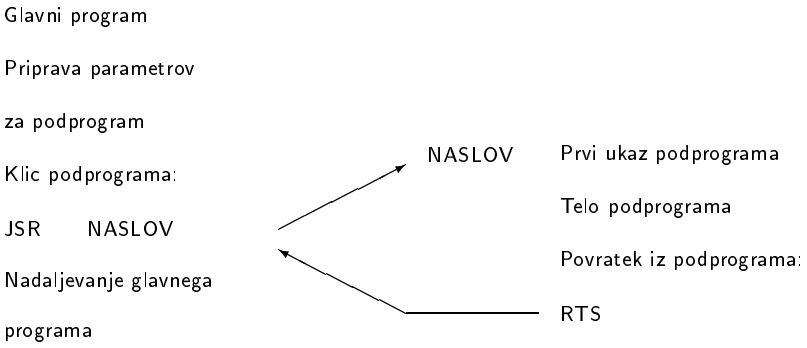
Vsebina besede z naslovom n-2 se kajpak ne spremeni, vendar ni več sestavni del sklada, ker skladovni kazalec po jemanju podatka iz sklada 'kaže' na naslednji podatek v skladu. Vsebina besede pa se spremeni ob naslednji operaciji VSTAVI. Pomni, zaporedje:

PSHA
PSHB
poljubna sprememba vsebine akumulatorjev
PULB
PULA

ohrani prvotno vsebino akumulatorjev. Takemu skladu pravimo tudi LIFO sklad (Last-In-First-Out Stack).

5.4 Povezovanje programov s podprogrami

Podprograme tipično pišemo tedaj, kadar se neko zaporedje ukazov v programu pogosto ponavlja. Sestavljanje večjih programov iz podprogramov, ki so po potrebi tudi sestavljeni iz podprogramov, nas navaja (sili) v strukturirano in modularno programiranje. 'Skok' v podprogram je običajno videti takole



Pri skoku v podprogram (JSR - Jump to SubRutine) se v sklad vstavi vsebina programskega števca, ki tedaj (že) vsebuje naslov naslednjega ukaza glavnega programa. V programski števec gre vsebina polja operanda, ki pomeni naslov prvega ukaza podprograma. Za ukaz:

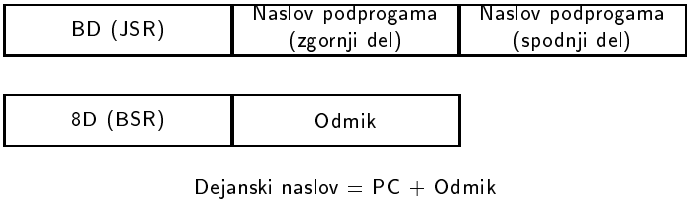
$$JSR \quad NASLOV$$

imamo naslednje zaporedje operacij:

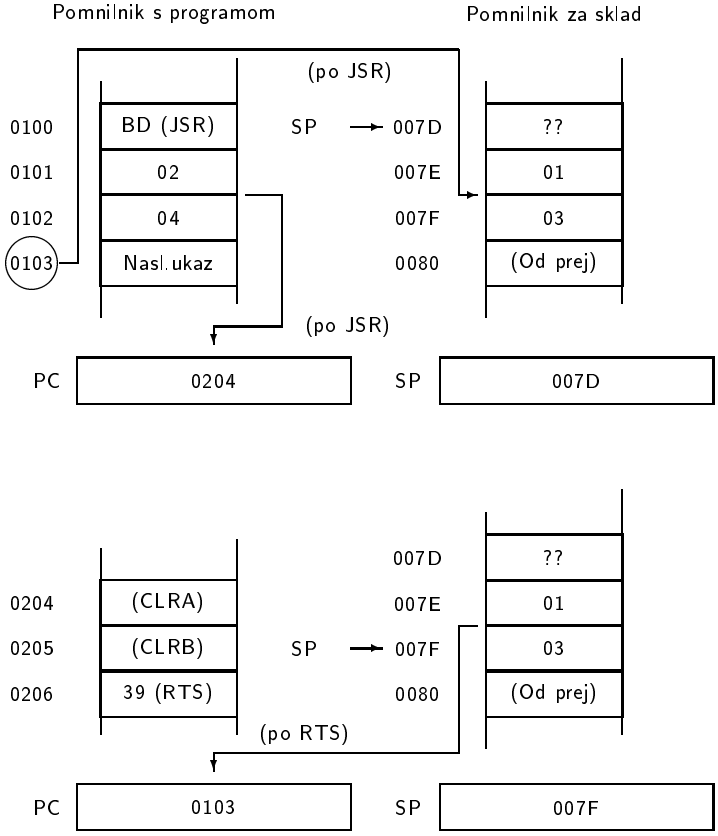
$$\begin{aligned}
&(SP) \leftarrow PC_L \\
&SP \leftarrow SP - 1 \\
&(SP) \leftarrow PC_H \\
&SP \leftarrow SP - 1 \\
&PC \leftarrow NASLOV
\end{aligned}$$

Zadnji ukaz, ki se mora izvršiti znotraj podprograma, je ukaz RTS (ReTurn from Subrutine), ki iz sklada obnovi vrednost programskega števca in glavni program se nadaljuje z ukazom, ki sledi ukazu JSR. Ukaz BSR (Branch to SubRutine) je podoben ukazu JSR, le da je 'skok' relativen glede na vsebino programskega števca - relativno naslavljanje.

Oblika ukaza (JSR, BSR):



Primer: Skok in povratek iz podprograma.



5.4.1 Primer programa s podprogramom

Naslednji zapis programa je podoben programu, na katerem smo razložili delovanje zbirnika, vendar vsebuje podprogram - subrutino.

```
        NAM     PRIMER
NPOD    EQU     10          ; število podatkov
*
*                           ; tu začne zaporedje ukazov
*
        ORG     $100        ; naslov prvega ukaza naj bo $100
        LDS     #SKLAD      ; začetni vrh sklada
*                           ; priprava parametrov za podprogram
        LDX     #ZNPOD      ; kazalec za naslavljanje podakov
        LDAB    #NPOD       ; števec za štetje podatkov
        JSR     SESTEJ      ; klic podprograma za seštevanje
TU      JMP     TU          ; in konec programa

*
*                             ; podprogram za seštevanje
*
SESTEJ  PSHA                  ; denimo, da hočemo ohraniti stare
        PSHB                  ; vrednosti akumulatorjev
        CLRA
ZANKA   ADDA    0,X           ; prištejemo vrednost podatka
        INX                   ; naslov naslednjega podatka
        DECB                  ; smo sešteli vse podatke?
        BNE     ZANKA         ; ne - prištej naslednjega
        STAA    0,X           ; da - shrani vsoto tik za tabelo podatkov
        PULB                  ; obnovimo vsebini akumulatorjev
        PULA
        RTS                   ; obvezno za pravilen povratek
*
*                             ; Prostor za sklad, podatke in rezultat
*
        RMB     20            ; prostor za sklad
SKLAD   EQU     * - 1         ; začetni vrh sklada
ZNPOD   RMB     NPOD+1        ; prostor za podatke in za rezultat
        END
```

5.5 Prekinitve in sklad

Mikroprocesor MC6800 ima **vektorski sistem prekinitev** za streženje zunanjim zahtevam za prekinitev in za streženje programskim zahtevam za prekinitev (SoftWare Interrupt).

V primeru zahteve od zunaj (na sponkah $\overline{NMI}$ in $\overline{IRQ}$) mikroprocesor dokonča tekoči ukaz, shrani vsebine vseh programsko dostopnih registrov v sklad (razen vsebine skladovnega kazalca), postavi maskirni bit I v registru stanj v stanje ena ($I = 1$), v programski števec pa naloži vsebino ustreznega vektorja zahteve za prekinitev. **Prekinitveni vektor** je naslov (dveh) pomnilniških besed glavnega pomnilnika z vnaprej predvidenim in

nespremenljivim naslovom. Torej, vsebina vektorja pomeni naslov prvega ukaza strežnega programa (Interrupt Service Routine), ki naj se izvede zaradi zahteve za prekinitev.

Dogajanje ob programski prekinitvi je podobno kot pri zahtevi za prekinitev od zunaj, vendar pride v tem primeru zahteva za prekinitev iz programa z ukazom SWI (SoftWare Interrupt).

Mikroprocesor ima dve vhodni sponki (skupaj s sponko $\overline{RESET}$ tri), na katerih ima druga naprava možnost postaviti zahtevo za prekinitev.

- Sponka $\overline{NMI}$ (NonMaskable Interrupt). Na tej sponki se zahteve za prekinitev ne da preprečiti z maskirnim bitom v registru stanj, do prekinitve pride ne glede na stanje bita I.
- Sponka $\overline{IRQ}$ (Interrupt Request). Na tej sponki se zahtevo za prekinitev da preprečiti. Do prekinitve ne pride, če je maskirni bit v stanju 1, $I = 1$.

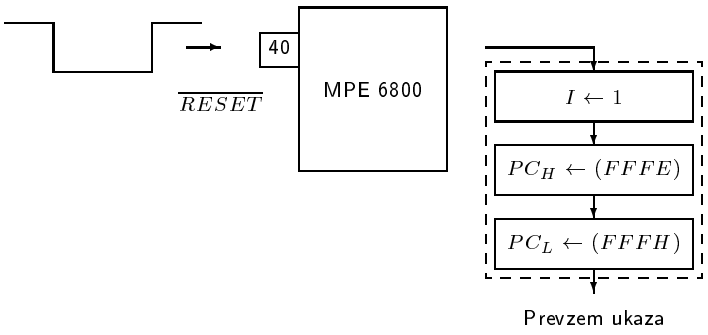
Vsak vhod ($\overline{RESET}, \overline{NMI}, \overline{IRQ}$, pa tudi ukaz SWI ima svoj vektor zahteve za prekinitev.

Vektor za začetni zagon mora vsebovati naslov prvega ukaza, ki naj se izvrši ob zagonu (vklopu) računalnika.

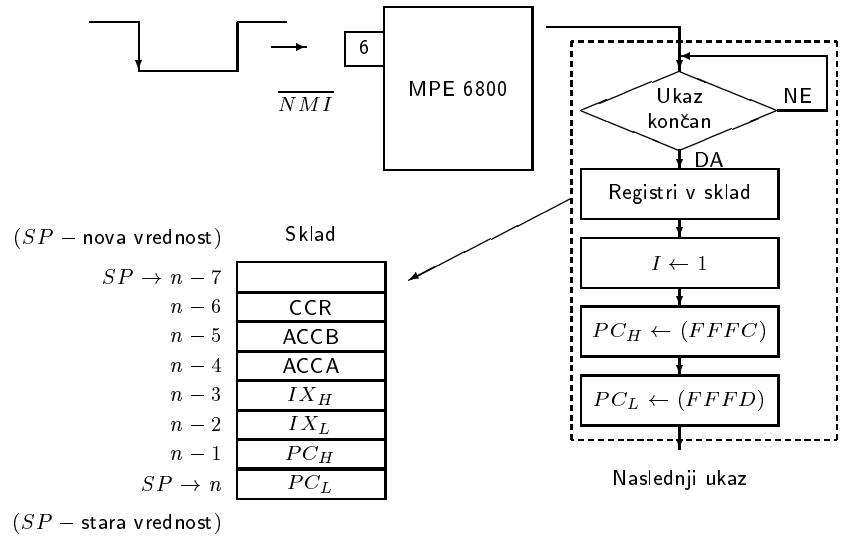
Zahtevo za prekinitev na sponki $\overline{NMI}$ pomeni aktivni prehod (od zgoraj navzdol) signala. Zahtevo za prekinitev na tej sponki običajno lahko postavi samo (obrobna) naprava s posebnim pomenom.

Zahtevo za prekinitev na sponki $\overline{IRQ}$ pomeni aktivno stanje signala. Zahteva za prekinitev obstaja, dokler je signal na tej sponki v aktivnem (nizkem) stanju. Na sponki $\overline{IRQ}$ postavljajo zahtevo za prekinitev obrobne naprave, ki so vezane preko vhodno izhodnih vmesnikov na računalnik.

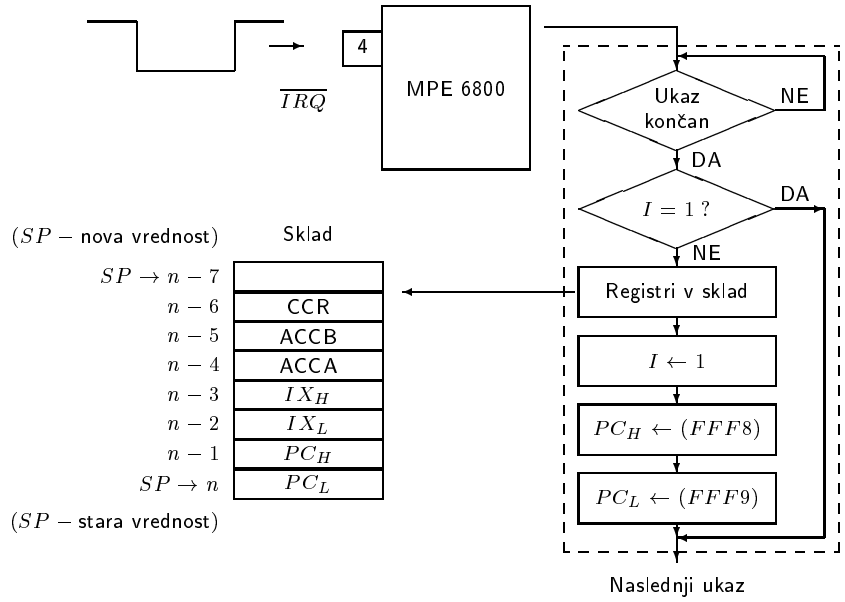
5.5.1 Začetni (ponovni) zagon (Reset)



5.5.2 Nemaskirana zahteva za prekinitev - na sponki NMI



5.5.3 Zahteva za prekinitev na sponki IRQ



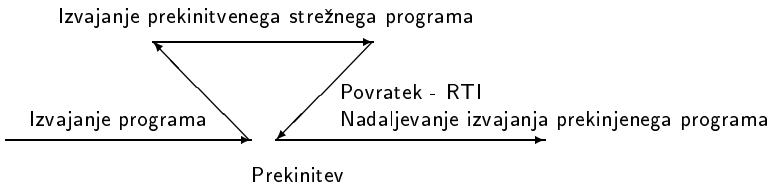
5.5.4 Vektorji zahtev za prekinitev

Vektorji zahtev za prekinitev zasedajo zadnjih osem pomnilniških besed v šestnajstbitnem naslovnem področju mikroprocesorja.

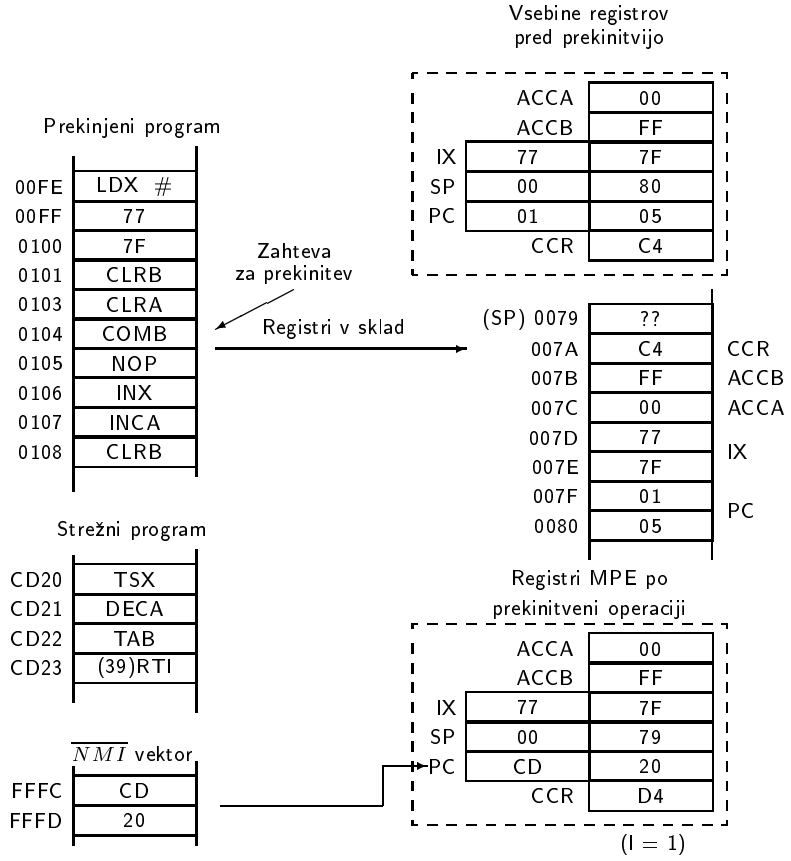
$\overline{IRQ}$ vektor	FFF8	$\overline{IRQ}$ (zgornji del)
	FFF9	$\overline{IRQ}$ (spodnji del)
$\overline{SWI}$ vektor	FFFA	$\overline{SWI}$ (zgornji del)
	FFFB	$\overline{SWI}$ (spodnji del)
$\overline{NMI}$ vektor	FFFC	$\overline{NMI}$ (zgornji del)
	FFFD	$\overline{NMI}$ (spodnji del)
$\overline{RESET}$ vektor	FFFE	$\overline{RESET}$ (zgornji del)
	FFFF	$\overline{RESET}$ (spodnji del)

5.5.5 Povratek iz prekinitvenega strežnika

Za povratek iz strežnega programa v prekinjeni program poskrbi ukaz RTI (ReTurn from Interrupt). Na koncu prekinitvenega strežnika **mora** biti torej obvezno ukaz RTI. Ukaz RTI obnovi iz sklada vsebine registrov mikroprocesorja (tja so bile shranjene v trenutku zahteve za prekinitev). S tem se v procesni enoti vzpostavi stanje tik pred prekinitvijo in prekinjeni program se nadaljuje na mestu prekinitve.



Primer: Streženje zahtevi za prekinitev na sponki $\overline{NMI}$.



Za povratek iz prekinitvenega strežnega programa v prekinjeni program poskrbi ukaz RTI. Ukaz RTI, ki se izvrši kot zadnji ukaz prekinitvenega strežnega programa, vzame iz sklada vsebine registrov, kamor so se shranile ob zahtevi za prekinitev in izvajanje prekinjenega programa se obnovi.

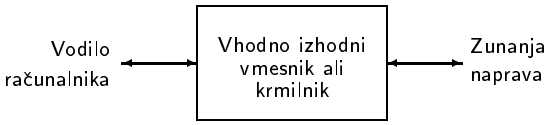
5.5.6 Pregled ukazov JSR, BSR, JMP, RTS in RTI

6 Povezovanje zunanjih naprav na računalnik

Zunanja naprava računalnika je naprava, ki pretvarja informacijo iz oblike, ki je primerna za človeka ali drugo napravo, v obliko, ki je primerna za obdelavo z računalnikom in obratno. Povezovanje zunanjih naprav na računalnik omogočajo vhodno izhodni vmesniki in krmilniki.

Vhodno izhodni (v/i) vmesnik je naprava, ki v električnem in v logičnem smislu prilagodi zunanjo napravo na vodilo računalnika.

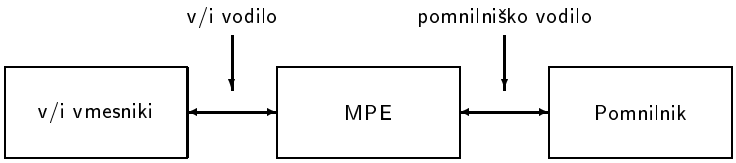
Vhodno izhodni krmilnik je zmogljivejši vmesnik, ki opravlja tudi nekatere (ali vse) osnovne operacije v zvezi z zunanjo napravo namesto procesne enote računalnika.



V majhnih računalnikih sta možna dva osnovna načina povezovanja v/i vmesnikov na vodilo:

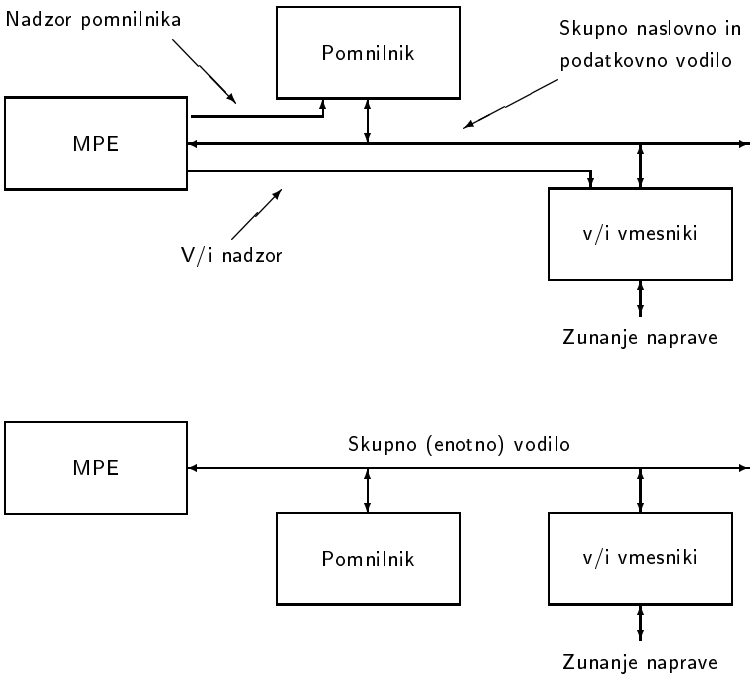
- v/i vodilo je ločeno od pomnilniškega vodila,
- vodilo za v/i vmesnike in pomnilnik je skupno (koncept skupnega vodila).

Slednji način je skoraj popolnoma izrinil prvega. Slika prikazuje računalnik z ločenim v/i vodilom.



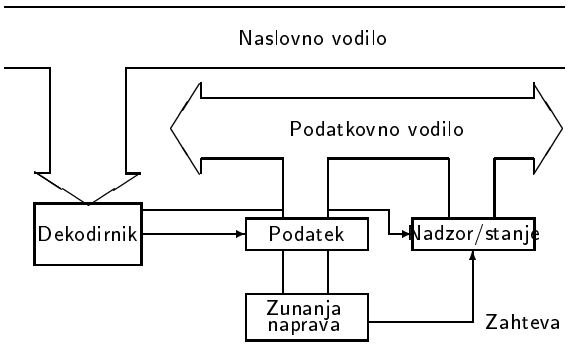
Pri računalnikih s skupnim vodilom obstajata dve različici:

- vhodno izhodno naslovno področje je ločeno od naslovnega področja pomnilnika. Procesna enota ima vhodno izhodne ukaze za delo z vmesniki. Za delo z vmesniki ima tudi posebne nadzorne signale (na primer signal, ki zahteva vhodno/izhodni prenos), medtem ko je v/i naslovno in v/i podatkovno vodilo skupno s pomnilniškim. Vmesniki so 'preslikani' v vhodno izhodno naslovno področje (Input/Output Mapped). Ta koncept je značilen za Intel-ove mikroprocesorje.
- Popolnoma enotno vodilo. S stališča procesne enote ni nobene razlike med pomnilnikom in vmesnikom. Nima niti ločenih v/i nadzornih signalov niti v/i ukazov. Vmesniki so preslikani v pomnilniško naslovno področje (Memory Mapped Input/Output). Procesna enota izvede vhodno izhodni prenos na enak način (z istimi ukazi) kot prenos iz ali v pomnilnik. Ta koncept je značilen za Motorolo in DEC, ki je začetnik koncepta enojnega vodila.



6.1 Enostaven vhodno izhodni vmesnik

Vmesnik mora zagotoviti *primerno pot* za prenos podatkov in ustrezna krmilna vezja za *nadzor* nad prenosom podatkov. Običajno vsebuje vsaj en podatkovni register, ki zadrži podatek, dokler ga ne prevzame procesna enota ali zunanja naprava, odvisno od smeri prenosa. Poleg podatkovnega registra običajno vsebuje še en register (kontrolni register), ki omogoča nadzor nad delovanjem vmesnika oziroma naprave in odraža njeno trenutno stanje.



6.2 Vhodno izhodni prenosi podatkov

Pod vhodno izhodnimi prenosi podatkov razumemo postopke za izvedbo prenosa podatkov od zunanje naprave (preko vmesnika) k računalniku in obratno.

Obstajata dva osnovna načina za izvedbo prenosa podatkov:

- s posredovanjem procesne enote (prenos opravi MPE) in
- brez posredovanja procesne enote (prenos opravi poseben krmilnik namesto MPE).

6.2.1 Prenos s posredovanjem procesne enote

Kot pove ime samo, opravi v tem načinu prenos podatkov procesna enota. Obstajata dve osnovni možnosti prenosa s posredovanjem procesne enote:

- s preverjanjem stanja zunanje naprave (oziroma vmesnika) (Polling) ali
- z zahtevo za prekinitev procesne enote s strani zunanje naprave.

V prvem načinu prevzame pobudo za prenos podatka procesna enota, v drugem načinu pride pobuda za prenos s strani zunanje naprave.

Preverjanje stanja

Procesna enota programsko (s programom) stalno ali občasno preverja stanje zunanje naprave (oziroma vmesnika). Če je izpolnjen pogoj za prenos podatka (od ali proti zunanji napravi), procesna enota opravi prenos. Tipično zaporedje ukazov v zbirniku za izvedbo tega načina je:

```
CAKAJ   LDAA    STANJE      ; preveri stanje zunanje enote
        CMPA    #POGOJ      ; je pogoj za prenos izpolnjen?
        BNE     CAKAJ       ; ne - preverjaj ponovno
        LDAA    PODATEK     ; da - prevzemi podatek
```

Dobri lastnosti tega načina sta:

- razmeroma visoka hitrost prenosa,
- enostavnost izvedbe v pogledu programske in materialne opreme.

Slabosti pa so:

- slaba izkoriščenost procesne enote,
- otežkočena pravočasna programska oskrba večjega števila zunanjih naprav,
- občutljivost na napake.

Prenos z zahtevo za prekinitev

Pobuda za prenos podatkov pride s strani zunanje naprave oziroma vmesnika, na katerega je naprava priključena. Dokler pogoji za prenos niso izpolnjeni, procesna enota rešuje tekoče probleme. Ko so pogoji za prenos izpolnjeni, zunanja naprava zahteva posredovanje procesne enote tako, da postavi zahtevo za prekinitev na enem od prekinitvenih

vhodov. Procesna enota dokonča tekoči ukaz, izvrši prekinitveno operacijo (shrani registre v sklad, i.t.d.), streže zunanji napravi v prekinitvenem strežnem programu, potem pa obnovi izvajanje prekinjenega programa.

Tipičen primer prenosa z zahtevo za prekinitev v računalniku z mikroprocesorjem MC6800 je:

```
VEKTOR  EQU   $FFF8      ; naslov prekinitvenega vektorja
ROM     EQU   $F000      ; naj bo tu programska oprema
RAM     EQU   $0000      ; naj bo tu delovni pomnilnik
*
        ORG   ROM        ; začetek programa
        LDS   #SKLAD     ; določimo sklad
        ...              ; sledi zaporedje ukazov glavnega programa
        CLI              ; od tu MPE dovoli prekinitve
        JMP   *          ; tu bi bil lahko poljuben program
*                        ; strežni program začne tu
INT     LDAA  PODATEK    ; prevzamemo podatek od zunaje naprave
        ...              ; sledijo ukazi strežnega programa
        RTI              ; povratek v prekinjeni program
*
        ORG   VEKTOR     ; to je za definicijo vektorja
        FDB   INT        ; prvi naslov strežnega programa
*
        ORG   RAM        ; to začne pomnilnik RAM
        RMB   $20        ; prostor za sklad
SKLAD   RMB   1
        ...
```

Prednosti prenosa z zahtevo za prekinitev so:

- boljša izkoriščenost procesne enote,
- manjša občutljivost na napake,
- možno je asinhrono streči večjemu številu zunanjih naprav.

Slaba stran je nekoliko manjša hitrost prenosa, ker se nekaj časa porabi za prekinitev in obnovitev izvajanja programa, nekoliko daljši odzivni čas in zahtevnejša materialna oprema.

6.2.2 Prenos brez posredovanja procesne enote

V tem načinu pride do prenosa podatkov med pomnilnikom in zunanjo napravo brez posredovanja procesne enote. Ta način je znan kot neposreden dostop do pomnilnika (Direct Memory Access ali DMA). Nalogo prenosa prevzame poseben krmilnik za neposreden dostop do pomnilnika, ki po potrebi od procesne enote zahteva, naj mu prepusti vodilo.

Takoj, ko je mogoče procesna enota prepusti nadzor nad vodilom krmilniku, se odlepi od vodila in krmilnik opravi prenos. Ko je prenos končan, krmilnik vrne vodilo procesni enoti.

6.3 Prenos med zunanjo napravo in vmesnikom/krmilnikom

Za prenos podatkov med zunanjo napravo in računalnikom (točneje vmesnikom oziroma krmilnikom naprave) obstajajo naslednje dve možnosti sinhronziacije oziroma usklajevanja:

- neusklajen način,
- usklajen način (hand-shaking),

Pri neusklajenem načinu se med zunanjo napravo in računalnikom prenašajo samo podatki brez nadzornih signalov: ena naprava odda podatek in druga ga prevzame. Med napravama se ne prenaša kontrolna informacija o tem, kdaj ena naprava odda podatek in kdaj, če sploh, ga druga prevzame.

V usklajenem načinu spremlja prenos podatkov prenos kontrolnih signalov. Načeloma poteka usklajevanje takole:

1. oddajna naprava zahteva od sprejemne naprave naj se priprava na sprejem podatkov,
2. sprejemna naprava mora pred začetkom prenosa to pripravljenost potrditi,
3. sledi prenos podatkov,
4. sprejemna naprava v nekaterih primerih potrdi sprejem podatkov.

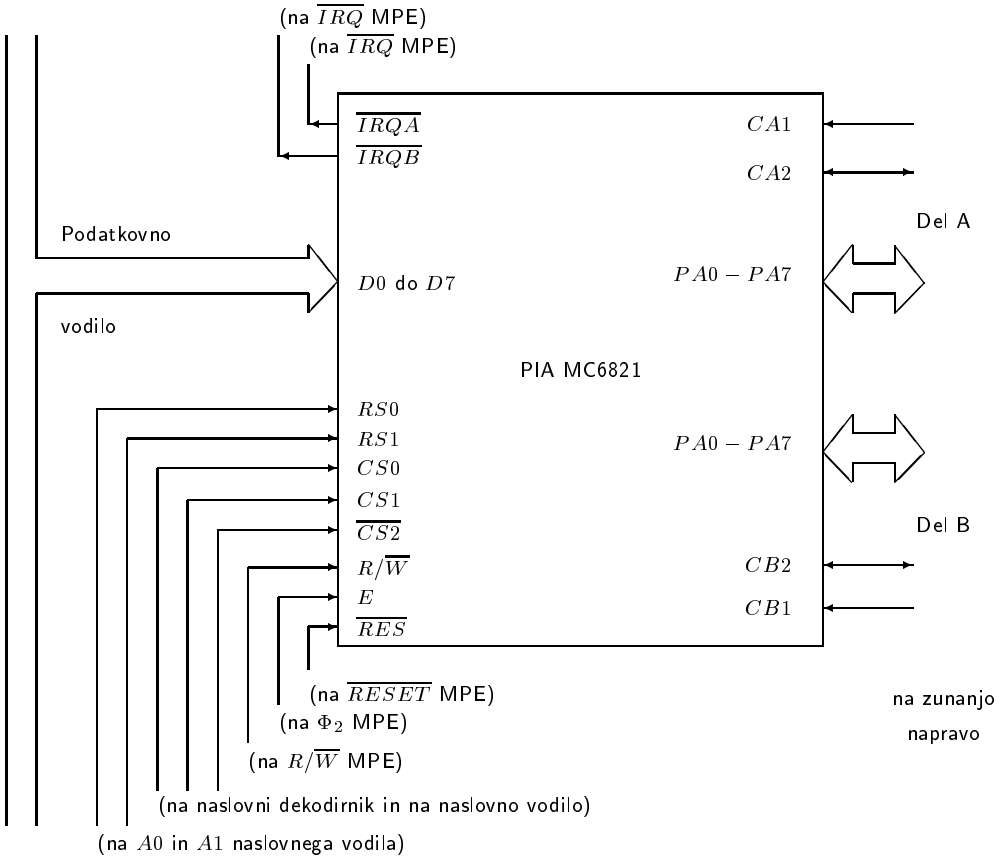
Obstaja več različic usklajevanja. Večkrat jih imenujemo tudi delno usklajen, usklajen in tudi dvojno usklajen. Pri (delno) usklajenem načinu Oddajna naprava pripravi podatek in ko je podatek pripravljen postavi še kontrolni signal. Sprejemna naprava zazna, da je kontrolni signal postavljen in prevzame podatek. V (polno) usklajenem načinu sprejemna naprava potrdi oddajni napravi prevzem podatka.

file=no-handshake.ps,width=50mm file=handshake.ps,width=50mm file=d-handshake.ps,width=50mm

7 Paralelni vmesnik PIA MC6821

Vmesnik MC6821 (Peripheral Interface Adapter), znan kot PIA, omogoča priključitev najrazličnejših zunanjih naprav na mikroračunalnik. Za povezavo na zunanjo napravo ima vmesnik dva para kontrolnih sponk za nadzor nad tokom podatkov med zunanjo napravo in mikroračunalnikom ter 16 podatkovnih sponk (dve skupini po osem), ki lahko delujejo kot vhodne ali kot izhodne sponke v poljubi kombinaciji.

Na sistemsko vodilo mikroračunalnika se priključi podobno kot pomnilniška vezja (ROM, RAM). Ima tri sponke za izbiro čipa ($CS0, CS1, \overline{CS2}$), dve sponki za izbiro enega od notranjih registrov ($RS0, RS1$), osem obojesmernih podatkovnih sponk za povezavo na podatkovno vodilo, sponko, ki postavi vmesnik v začetno stanje ($\overline{RES}$), dve prekinitveni sponki ($\overline{IRQA}, \overline{IRQB}$), sponko beri/piši ($R/\overline{W}$) in sponko za sinhronizacijo z dugimi vezji na vodilu (E - Enable).



7.1 Priključitev vmesnika PIA na sistemsko vodilo

7.1.1 Krmilni signali - priključitev na kontrolno vodilo

- **E (Enable) (vhod)**: sinhronizacija delovanja vmesnika z ostalimi napravami na

vodilu s sitemsko uro, običajno signalom ure Φ_2.

- $R/\overline{W}$ **(Read/Write) (Beri/Piši) (vhod)**: Signal, ki določa smer prenosa podatkov na podatkovnih sponkah $D0$ do $D7$. Visok nivo (beri): iz vmesnika, nizek nivo (piši): v vmesnik.
- $\overline{RES}$ **(Reset) (vhod)**: Signal, ki postavi vmesnik v začetno stanje (vsebine notranjih registrov vmesnika so nič).
- $\overline{IRQA}, \overline{IRQB}$ **(izhod z odprtim kolektorjem)**: izhoda zahteve za prekinitev mikroprocesorja za del A ($\overline{IRQA}$) in za del B ($\overline{IRQB}$), običajno vezana na sponko mikroprocesorja $\overline{IRQ}$.

7.1.2 Izbiranje vmesnika - naslovni signali

- $CS0, CS1, \overline{CS2}$ **(vhodi)**: izbira vmensika PIA, običajno so vezani preko dekodirnika na naslovno vodilo.
- $RS0, RS1$ **(vhoda)**: izbira enega od (šestih) notranjih registrov vmesnika, običajno sta vezana na naslovne sponke vodila (RS0 na A0 in RS1 na A1).

7.1.3 Podatkovne sponke - na podatkovno vodilo

- $D0$ **do** $D7$ **(obojesmerne)**: osem podatkovnih sponk za priključitev na podatkovno vodilo.

7.2 Sponke za priključitev na zunanjo napravo

7.2.1 Podatkovne sponke

- **PA0** do **PA7** (vhodi ali izhodi): osem podatkovnih sponk dela A. Posamično se jih da programirati za vhod ali za izhod.
- **PB0** do **PB7** (vhodi ali izhodi): osem podatkovnih sponk dela B. Posamično se jih da programirati za vhod ali izhod. Izhodni gonilniki so tristanjski.

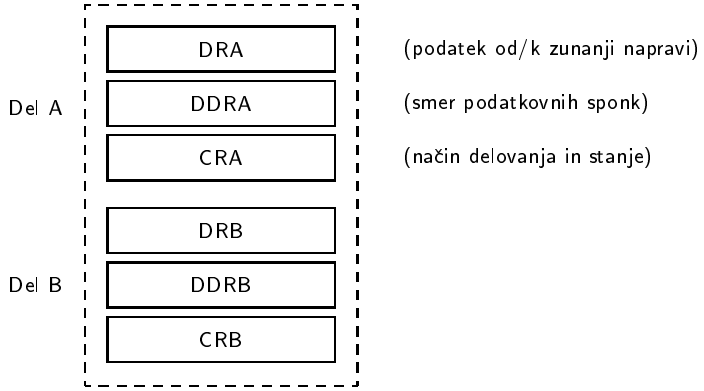
7.2.2 Kontrolne sponke

- **CA1, CB1** (vhoda): vhoda zahteve po oskrbi zunanje naprave dela A ($CA1$) in dela B ($CB1$).
- **CA2, CB2** (vhoda ali izhoda): kontrolni sponki programirani za vhod ali za izhod. Kot vhod: za zahtevo po oskrbi zunanje naprave, kot izhod: za krmiljenje zunanje naprave.

7.3 Programski model vmesnika

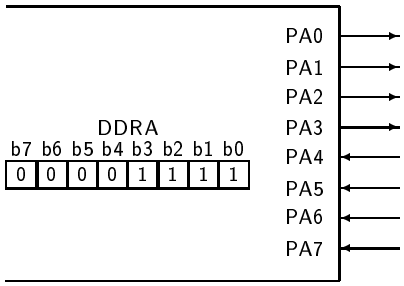
Vmesnik sestavljata dva neodvisna in s stališča programerja (skoraj) enakovredna dela: del A in del B. S stališča programerja ima vsak od obeh delov vmesnika tri (notranje) registre:

- podatkovni register (označujemo ga z DRx, x = A ali B),
- smerni register (označujemo ga z DDRx, x = A ali B),
- kontroni register (označujemo ga s CRx, x = A ali B).



Vsebina smernih registrov DDRA in DDRB določa smer perifernih podatkovnih sponk PA0 do PA7 in PB0 do PB7. Enica pomeni, da je istoležna podatkovna sponka programirana za izhod. Ničla pomeni, da je ustrezna sponka določena za vhod. Možna je poljubna kombinacija vhodov in izhodov.

Primer: naj so sponke PA0 do PA3 izgodne, sponke PA4 do PA7 pa vhodne. Vsebina smernega registra DDRA mora biti potem:

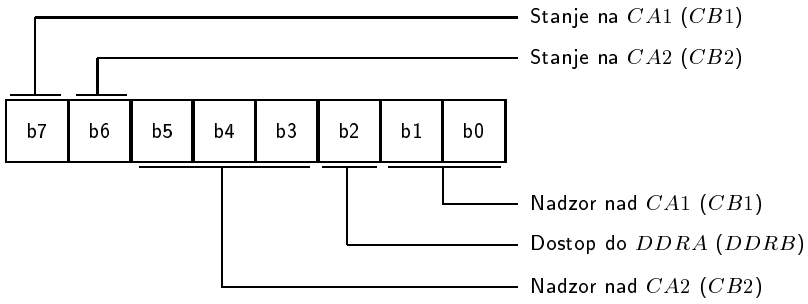


Podatkovna registra DRA in DRB služita za prenos podatkov med zunanjo napravo in sistemskim vodilom. Mikroprocesor lahko v ta register vpiše vsebino in ta vsebina je prisotna na perifernih podatkovnih sponkah, če so programirane za izhod. Če so periferne sponke programirane za vhod, mikroprocesor prebere stanje na podatkovnih sponkah z branjem tega registra.

Vsebina v kontrolnih registrih (CRA in CRB) določa način delovanja vmesnika (za vsak del posebej) in prikazuje trenutno stanje vmesnika. Biti b0 do b5 so čitljivi in vpisljivi, bita b6 in b7 sta samo čitljiva in (prikazujeta stanje vmesnika).

7.4 Pomen vsebine kontrolnega in statusnega registra

Pomen bitov kontrolnega regista CRA je skoraj enak pomenu bitov kontrolnega registra CRB, zato bomo oba registra obravnavali sočasno.



Bit b2 (DDR bit) (Data Direction Register Access bit) omogoča dostop (naslavljanje) smernega registra, skupaj s signaloma na sponkah RS0 in RS1.

RS1	RS0	CRA DDRA bit	CRB DDRB bit	Izbrani register
0	0	0	x	DDRA - Smerni register dela A
0	0	1	x	DRA - Podatkovni register dela A
0	1	x	x	CRA - Kontrolni register dela A
1	0	x	0	DDRB - Smerni register dela B
1	0	x	1	DRB - Podatkovni register dela B
1	1	x	x	CRB - Kontrolni register dela B

Podatkovni in smerni register imata na naslovnem vodilu isti naslov, dostop do smernega regisitra pa omogoča bit b2 v kontrolnem registru. V začetni (ali vsaki ponovni) pripravi vmesnika postavimo (programsko) ta bit v stanje 0. S tem postane dosegljiv smerni register. Ko vanj vpišemo ustrezen vzorec ničel in enic (izberemo smer podatkovnih sponk), postavimo bit b2 v kontrolnem registru v stanje 1. Od tedaj naprej je z istim naslovom na vodilu dosegljiv podatkovni register. Med obratovanjem vmesnika ostane ta bit ves čas v stanju ena.

Biti b0, b1 in b7 se nanašajo na kontrolni vhod CA1 (CB1). Bit b1 določa aktiven prehod signala na tej sponki (prvi ali zadnji rob impulza), bit b0 omogoči ali onemogoči zahtevo za prekinitev na sponki $\overline{IRQA}$ ($\overline{IRQB}$) ob aktivnem prehodu na sponki $CA1$ ($CB1$).

b1	b0	Ahtiven prehod na $CA1$ ($CB1$)	Zahteva za prekinitev na izhodu $\overline{IRQA}$ ($\overline{IRQB}$)
0	0	$\downarrow$	onemogočena
0	1	$\downarrow$	omogočena
1	0	$\uparrow$	onemogočena
1	1	$\uparrow$	omogočena

Ob aktivnem prehodu signala na sponki $CA1$ ($CB1$) se postavi bit b7 kontrolnega registra CRA (CRB) v stanje 1. Če je prekinitev na sponki $\overline{IRQA}$ ($\overline{IRQB}$ omogočena (z bitom b0 = 1), gre tudi signal na tej sponki v aktivno (nizko) stanje. Z **branjem** podatka iz podatkovnega registra DRA (DRB) se bit b7 vrne v stanje 0, prekinitveni izhod pa se vrne v neaktivno (visoko) stanje.

Biti b6, b5,b4 in b3 se nanašajo na sponko $CA2$ ($CB2$). Ko je bit b5=0, je kontrolna sponka namenjena za vhod. Bit b3 omogoči ali onemogoči zahtevo za prekinitev na sponki $\overline{IRQA}$ ($\overline{IRQB}$) ob aktivnem prehodu na sponki $CA2$ ($CB2$). Aktiven prehod signala na tej sponki je določen s stanjem bita b4.

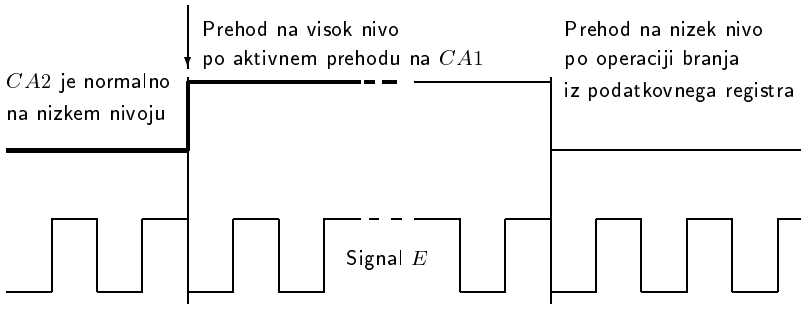
b5	b4	b3	Aktiven prehod na $CA2$ ($CB2$)	Izhod $\overline{IRQA}$ ($\overline{IRQB}$)
0	0	0	$\downarrow$	onemogočen
0	0	1	$\downarrow$	omogočen
0	1	0	$\uparrow$	onemogočen
0	1	1	$\uparrow$	omogočen

Ko je bit b5=1, deluje kontrolna sponka $CA2$ ($CB2$) kot izhodna. Obnašanje sponke $CA2$ se nekoliko razlikuje od obnašnaja sponke $CB2$.

7.4.1 Nadzor $CA2$: kombinacija b5=1, b4=0, b3=0

Tak način delovanja pride prav tedaj, kadar računalnik sprejema podatke od zunanje naprave. Na sponki $CA2$ računalnik potrdi sprejem podatka. Tipično zaporedje operacij:

- zunanja naprava postavi podatek na sponke $PA0$ do $PA7$ in javi prisotnost podatka na sponki $CA1$.
- Signal na $CA2$ gre zaradi prehoda signala na sponki $CA1$ iz nizkega v visoko stanje.
- Ko mikroprocesor opravi branje podatka iz podatkovnega registra, se signal na $CA2$ vrne iz visokega v nizko stanje.



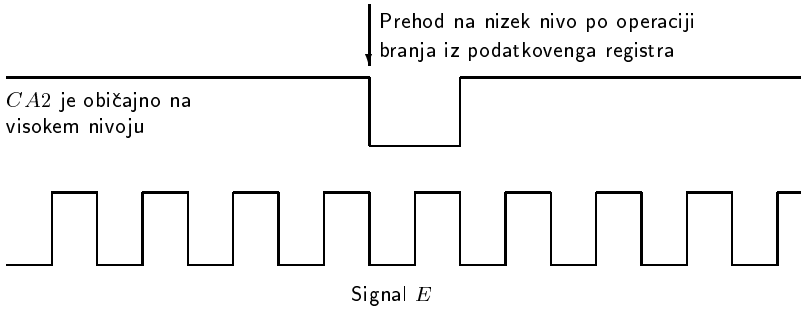
7.4.2 Nadzor $CB2$: kombinacija b5=1, b4=0, b3=0

Tak način delovanja pride prav tedaj, kadar računalnik pošilja podatke proti zunanji napravi. Obnašnaje signala na sponki $CB2$ je podobno obnašanju signala na sponki $CA2$, le da se sponka $CB2$ vrača v neaktivno stanje z operacijo vpisa (in ne branja) v podatkovni register vmesnika.

7.4.3 Nadzor $CA2$: kombinacija b5=1, b4=0, b3=1

Pri tem načinu delovanja računalnik javlja zunanji napravi, da je podatek prevzet. Uporablja se pri sprejemanju podatkov. Tipično zaporedje operacij:

- vhodna naprava pripravi podatek in to javi na sponki $CA1$.
- Branje podatka iz podatkovnega registra povzorči impulz na sponki $CA2$.



7.4.4 Nadzor $CB2$: kombinacija b5=1, b4=0, b3=1

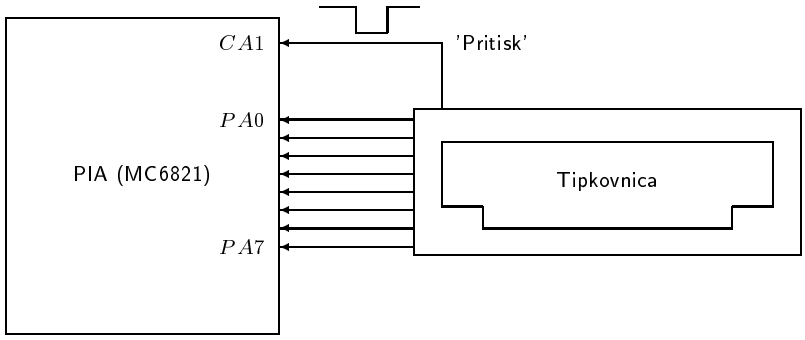
Tak način delovanja pride prav tedaj, kadar računalnik pošilja podatke proti zunanji napravi. Obnašanje signala na sponki $CB2$ je podobno obnašanju signala na sponki $CA2$, le da se pojavi impulz na sponki $CB2$ z operacijo vpisa (in ne branja) v podatkovni register vmesnika.

7.4.5 Nadzor $CA2$ ($CB2$): kombinacija b5=1, b4=1, b3=x

Ti dve kombinaciji dajeta možnost neposrednega programskega nadzora nad stanjem sponke $CA2$. Stanje sponke $CA2$ sledi stanju bita b3. Če je bit b3=0, je stanje na sponki $CA2$ nizko. Za b=1 je sponka $CA2$ v visokem stanju.

7.5 Enostavnejši primer uporabe vmesnika

Na računalnik (A del vmesnika PIA) priključimo tipkovnico. Ob pritisku tipke na tipkovnici se na podatkovnih izhodih pojavi podatek, nadzorni izhod 'Pritisk' pa gre z visokega na nizek nivo in ostane na nizkem nivoju za nekaj milisekund.



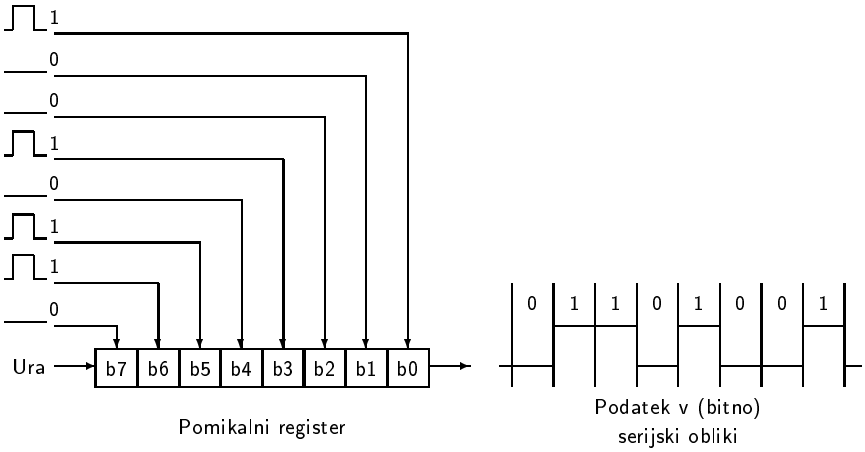
Zapišimo kos programa v zbirnem jeziku, ki pripravi vmesnik PIA in nato sprejme podatek s tipkovice.

```
PIADRA  EQU   $ 8000      ; naslov podatkovnega registra
PIACRA  EQU   PIADRA+1    ; naslov kontrolnega registra
        ...
*                         ; najprej priprava vmesnika
        CLRA
        STAA  PIACRA      ; b2=0 in dosegljiv je DDRA
        LDAA  #$04        ; b2=1 in dosegljiv je DRA
        STAA  PIACRA      ; IRQA onemogočen, CA1 aktiven ↓
        ...
*                         ; sprejem podatka
ZANKA   LDAA  PIACRA      ; zanima nas bit b7
        BPL   ZANKA       ; b7 = 0, ni podatka
        LDAA  PIADRA      ; b7 = 1, beremo podatek in b7 gre v 0
        ...
```

7.6 Podrobnejši opis vsebine kontrolnega registra

8 Asinhroni serijski vmesnik ACIA MC6850

Glavna naloga asinhronega serijskega vmesnika MC6850 (Asynchronous Communications Interface Adapter), bolj znanega pod kratico ACIA, je pretvorba podatkov iz (bitno) paralelne oblike (take, kot je na podatkovnem vodilu računalnika) v (bitno) serijsko obliko (tako, ki jo zahteva večina današnjih komunikacijskih naprav). Vmesnik torej omogoča razmeroma enostavno priključevanje komunikacijskih naprav na računalnik. Poleg paralelno serijske in serijsko praralene pretvorbe pa opravlja tudi važnejše nadzorne funkcije, ki so potrebne pri asinhronem serijskem načinu komuniciranja.



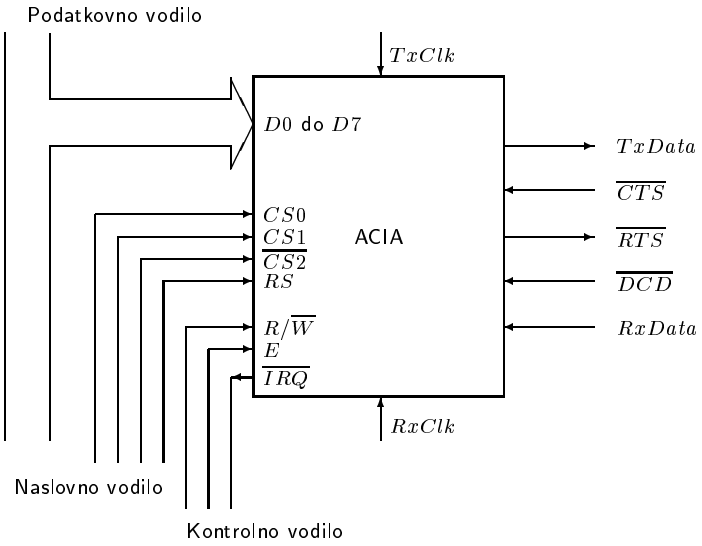
8.1 Opis sponk oziroma signalov

Sponke vmesnika v grobem delimo na tiste, ki so za priključitev vmesnika na sistemsko vodilo in na tiste, ki jih priključimo na obrobno napravo.

8.1.1 Priključitev vmesnika na sistemsko vodilo

- **CS0**, **CS1**, $\overline{\textbf{CS2}}$ (Chip Select) (vhodi): Sponke za izbiranje vmesnika, ki jih vežemo preko dekodirnika naslovov na naslovno vodilo.
- **RS** (Register Select) (vhod): Izbiranje enega ali drugega para notranjih registrov vmesnika.
- **D0** do **D7** (TS, obojesmerne): sponke za priključitev na podatkovno vodilo.
- **E** (Enable) (vhod): sinhronizacija vmesnika z ostalimi vezji na vodilu. Sponka je običajno vezana na signal Φ_2.
- **R**/$\overline{\textbf{W}}$ (Read/Write) (vhod): Sponka, ki določa smer prenosa podatkov in hkrati izbere enega od dveh notranjih registrov vmesnika. Beri: podatek od vmesnika na vodilo; piši: podatek z vodila k vmesniku.

- **$\overline{\text{IRQ}}$** (Interrupt Request) (izhod): zahteva za prekinitev MPE s strani vmesnika ACIA; običajno je vezana na vhod **$\overline{\text{IRQ}}$** mikroprocesorja.



8.1.2 Priključitev na zunanjo napravo

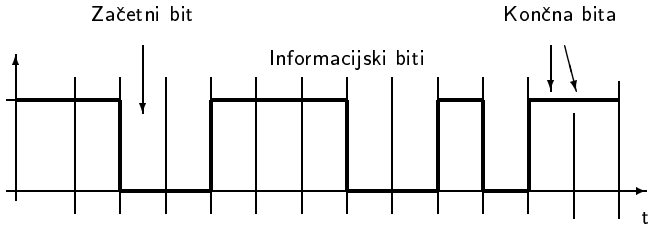
Za priključitev na obrobno napravo imamo dve podatkovni sponki: sponko za oddajo podatkov in sponko za sprejem podatkov v asinhroni serijski obliki. Tri nadzorne sponke služijo za nadzor nad tokom podatkom med vmesnikom ACIA in MODEMom (Modulator - Demodulator), možno pa jih je koristiti tudi pri povezavi z drugimi napravami (terminali, drugimi računalniki, ...).

- **RxData** (Receive Data) (vhod): sponka za sprejem podatkov, ki prihajajo v serijski obliki od periferne naprave.
- **TxData** (Transmit Data) (izhod): sponka za oddajo podatkov v serijski obliki proti periferni napravi.
- **$\overline{\text{CTS}}$** (Clear To Send) (vhod): sponka, na kateri MODEM javlja, da se je pripravil na oddajo, torej je pripravljen sprejeti podatke od vmesnika ACIA na sponki $TxData$.
- **$\overline{\text{RTS}}$** (Request To Send) (izhod): sponka, na kateri vmesnik ACIA postavi MODEMu zahtevo, naj se pripravi na oddajo.
- **$\overline{\text{DCD}}$** (Data Carrier Detect) (vhod): sponka, na kateri MODEM javlja, da so podatki, ki jih pošilja vmesniku ACIA po sponki $RxData$, veljavni.

Vmesnik ACIA ima še dve vhodni sponki za priključitev generatorja pravokotnih impulzov (ure). Signal na sponki **TxClk** določa hitrost oddajanja serijskega zaporedja bitov na sponki $TxData$. Signal na sponki **RxClk** določa hitrost sprejemanja serijskega zaporedja podatkov na sponki $RxData$.

8.2 Asinhrona serijska oblika podatkov

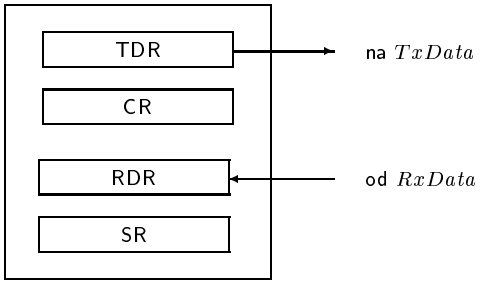
Pri asinhronem serijskem prenosu podatkov se sinhronizacija sprejemnika z oddajnikom opravlja za vsak posamezen podatek (tipično osem bitov). Vsak podatek v zaporedju podatkov ima svoje sinhronizacijske bite. Oddajna naprava podatku od 'spredaj' pripne začetni (start) bit in od 'zadaj' ga zaključi s končnim (stop) bitom ali dvema končnima bitoma. Premor med zaporednimi podatki sme biti poljubno dolg. Podatki torej lahko prihajajo asinhrono v nepredvidljivo dolgih časovnih presledkih. Sprejemna naprava se sinhronizira z oddajno napravo ob sprejemu začetnega bita, nato pa odbira podatkovne bite v trenutkih, kot jih določa takt njene (lokalne) sprejemne ure.



Da je sprejemanje podatkov pravilno, se mora hitrost sprejemnika ujemati s hitrostjo oddajnika, dogovorjeno pa mora tudi število bitov na podatek in število končnih bitov oddajnika in sprejemnika. Končni biti so potrebni za izravnavo razlike med hitrostjo oddajnika in hitrostjo sprejemnika. Pravzaprav določajo najkrajši možni časovni presledek med dvema zaporednima oddanima podatkomoma, sicer pa je presledek poljubno daljši.

8.3 Notranji registri vmesnika

Vmesnik ACIA vsebuje štiri programsko dosegljive registre: oddajni podatkovni register (TDR), sprejemni podatkovni register (RDR), kontrolni register (CR) in register stanja (SR).



V oddajni podatkovni register in v kontrolni register se da samo vpisati vsebino, ne da pa se je brati. Iz sprejemnega podatkovnega registra in iz registra stanja pa je možna operacija branja.

RS	$R/\overline{W}$	Izbrani register
0	0	Kontrolni register (CR)
0	1	Register stanja (SR)
1	0	Oddajni register (TDR)
1	1	Sprejemni register (RDR)

Sponka za izbiro registra je običajno vezana na sponko $A0$ naslovnega vodila. Kontrolni register in register stanja imata na vodilu enak naslov. Podobno imata enak naslov oddajni in sprejemni register. ACIA zaseda v naslovnem področju pomnilnika torej dva naslova.

Z vpisom ustreznega bitnega vzorca v kontrolni register določimo način delovanja vmesnika. Z branjem vsebine registra stanja ugotovimo trenutno stanje vmesnika (sprejet je podatek, oddaja podatka je možna, pri sprejemanju podatka je nastopila napaka, i.t.d.).

Podatek (osem bitov), ki ga vpišemo v oddajni register, se spremeni v asinhrono serijsko obliko in odda na sponki $RxData$. Podatek, ki pride v asinhroni serijski obliki na sponki $RxData$, se spremeni v paralelno obliko in možno ga je prebrati iz sprejemnega registra.

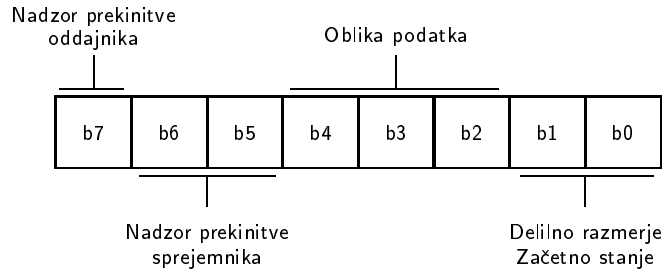
Vmesnik ACIA sam dodaja in izloča začetne (Start) in končne (Stop) bite ter preverja pravilnost prenosa podatkov med vmesnikom in obrobno napravo.

8.4 Programiranje vmesnika

Predno začnemo prenašati podatke, moramo določiti način delovanja vmesnika. To dosežemo z vpisom ustreznega vzorca bitov v kontrolni register. Tako določimo obliko serijskega podatka: število podatkovnih bitov, dodajanje in preverjanje parnostnega bita, število končnih bitov ter delilno razmerje signala oddajne in sprejemne ure.

Še prej moramo vmesnik postaviti v začetno stanje. Vmesnik ACIA nima sponke za signal $\overline{RESET}$, ki bi ga postavil v začetno stanje, kot recimo vmesnik PIA. Začetno stanje vmesnika se doseže programsko z vpisom vrednosti 3 v kontrolni register (bita b0 in b1 postavimo v ena).

8.4.1 Pomen vsebine kontrolnega registra



Izbiranje delilnega razmerja in postavljanje vmesnika v začetno stanje:

b1	b0	delilno razmerje urinih impulzov
0	0	1
0	1	16
1	0	64
1	1	Začetno stanje

Izbiranje dolžine besede, preverjanja parnosti in določanje števila končnih bitov:

b4	b3	b2	Dolžina podatka	Preverjanje parnosti	Končni biti
0	0	0	7	sodo	2
0	0	1	7	liha	2
0	1	0	7	soda	1
0	1	1	7	liha	1
1	0	0	8	brez	2
1	0	1	8	brez	1
1	1	0	8	soda	1
1	1	1	8	liha	1

Z bitoma b6 in b5 omogočimo ali onemogočimo zahtevo za prekinitev na sponki $\overline{IRQ}$ vmesnika ACIA tedaj, ko je oddajni register prazen (podatek oddan). S tema bitoma nadzorujemo tudi stanje sponke $\overline{RTS}$ (zahteva za oddajo).

b6	b5	$\overline{RTS}$	$\overline{IRQ}$
0	0	0	onemogočen
0	1	0	omogočen
1	0	1	onemogočen
1	1	0	onemogočen (Premor 'Break')

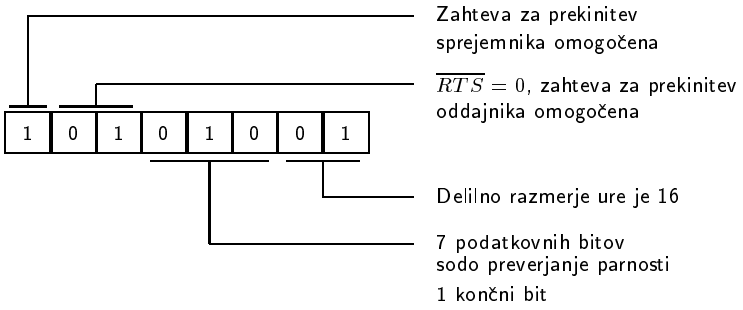
Bit b7 omogoči prekinitev na sponki $\overline{IRQ}$ pri sprejemu (tedaj, ko je podatkovni register poln - podatek sprejet).

b7	sponka $\overline{IRQ}$
0	prekinitev onemogočena
1	prekinitev omogočena

8.4.2 Primer začetne priprave vmesnika

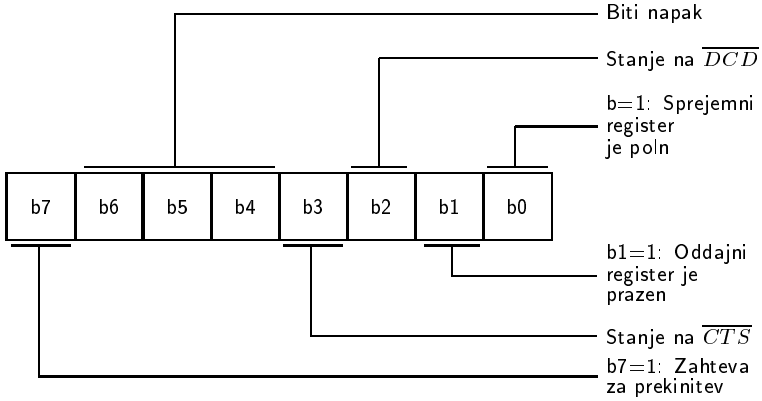
```
                                ;Simbolični imeni naslovom vmesnika
ACIACS   EQU    $8000           ;naslov kontrolnega in statusnega registra
ACIADR   EQU    ACIACS+1        ;naslov oddajnega in sprejemnega registra

         LDAA   #03             ;postavimo vmesnik v začetno stanje
         STAA   ACIACS

         LDAA   #$A9            ;začetna priprava vmesnika
         STAA   ACIACS
```

S tem dosežemo:



8.4.3 Pomen vsebine v registru stanja

Vsebina registra stanja odraža trenutno stanje vmesnika.



b0	Stanje sprejemnega podatkovenga registra
0	register je prazen (podatka še ni)
1	register je poln (podatek je prisoten in možno ga je prebrati

b1	Stanje oddajnega podatkovnega registra
0	register je poln - oddaja podatka je v teku
1	register je prazen in možen je vpis naslednjega podatka

b2	Preverjanje stanja vhoda $\overline{DCD}$
0	vhod je aktiven (nizek) - nisilec je prisoten
1	vhod ni aktiven (visok) - nosilec ni prisoten

b3	Stanje vhoda $\overline{CTS}$
0	vhod je v aktivnem stanju (nizek) - oddaja je možna
1	vhod ni v aktivnem stanju - oddaja je onemogočena

Biti b4, b5, in b6 so biti napak. Stanje 1 pomeni prisotnost napake pri sprejemu podatka in podatek je v tem primeru neveljaven.

b6	b5	b4	Vrsta napake
x	x	1	napaka v sinhronizaciji
x	1	x	prehitevanje podatkov
1	x	x	napaka parnosti

b7	Zahteva za prekinitev na sponki $\overline{IRQ}$
0	ni zahteve
1	je zahteva

8.4.4 Primer za sprejem podatka

Bit b0 v registru stanja upoštevamo pri sprejemanju podatkov. Podatek pride na sponki *RxData* ter se pretvori v paralelno obliko. Njegova vrednost je prisotna v sprejemnem podatkovnem registru, bit b0 v registru stanja pa gre v stanje 1. S preverjenjem tega bita procesna enota ugotovi, da lahko prebere podatek. Z branjem podatka iz sprejemnega registra se bit b0 vrne v stanje 0.

```
ZANKA   LDAA    ACIACS
        BITA    #01         ;zanima nas bit b0
        BEQ     ZANKA       ;b0 = 0 in podatka še ni
        BITA    #$70        ;zanimajo nas biti napak
        BNE     NAPAKA      ;ena od napak
        LDAA    ACIADR      ;sprejmemo podatek
        ....
```

Druga možnost sprejemanja podatkov je sprejemnaje podatkov z omogočeno zahtevo za prekinitev ob sprejemu. Ko je podatek sprejet in prisoten v sprejemnem registru, gre izhod $\overline{IRQ}$ vmesnika v aktivno (nizko) stanje. To za mikroprocesor pomeni zahtevo, naj izvrši strežni program zahteve za prekinitev, v katerem prebere podatek.

8.4.5 Primer za oddajo podatka

Pri oddaji podatkov moramo upoštevati bit b1 v registru stanja vmesnika. Ko je bit b1 v stanju 1, smemo v oddajni register vpisati podatek. Ko podatek vpišemo v oddajni register, gre ta bit v stanje 0. Ko se podatek prepiše v register za pretvorbo v serijsko obliko, je oddajni register ponovno 'prazen', bit b1 gre v stanje 1 in možno je vpisati (oddati) naslednji podatek.

```
CAKAJ   LDAB    ACIACS
        BITB    #02         ;zanima nas bit b1
        BEQ     CAKAJ       ;b1 = 0 in oddaja ni možna
        STAA    ACIADR      ;oddamo podatek
```

8.4.6 Kontrolni register - podrobno

8.4.7 Register stanja - podrobno

9 Mikrokrmilniki družine MC6801

Mikrokontrolerji družine MC6801/MC6803 imajo za osnovo izpopolnjen mikroprocesor MC6800, ki je z njim navzgor združljiv na nivoju strojnega jezika. Akumulatorja A in B sta osembitna ali pa skupaj sestavljata 16-biten akumulatorski par označen z D (D = A:B, A za zgornji del, B za spodnji del). MC6801 ima izboljšan in razširjen nabor ukazov mikroprocesorja MC6800:

- zmore osembitno nepredznačeno (8×8) množenje: (ukaz MUL, ki zmnoži vsebini v A in B in da 16-bitni rezultat v D),
- ima nekatere nove šestnajstbitne prenosne ter aritmetične in logične operacije (z akumulatorjem D): LDD, STD, ADDD, SUBD, ASLD, LSLD, LSRD,
- možno je shranjevanje indeksnega registra v sklad: PSHX, PULX, ter prištevanje akumulatorja B k regsitru X: ABX,
- nove vejitvene ukaze: BHS, BLO, BRN, ter izpopolnjena ukaza CPX (možen z vsakim vejitvenim ukazom), in JSR (direkten način naslavljanja).

Mikrokontroler MC6801 ima lasten generator urinega signala, ima vgrajen serijski komunikacijski vmesnik (SCI) in šestnajstbiten programljiv časovnik (Timer). Ima štiri paralelne vmesnike (vrata), v seštevku 29 sponk, ki se jih da programirati za vhod ali za izhod, za vrata 3 pa še dve nadzorni sponki (SC1 in SC2). Ima notranji pomnilnik RAM (128 bajtov tipično) z možnostjo stalnega baterijskega napajanja (spodnjih 64 bajtov) ter notranji pomnilnik tipa ROM izbrane velikosti. Vsebino pomnilnika ROM določi proizvajalec v času izdelave po naročilu uporabnika. Zato je ekonomičen za večjo količino enakih elementov (s programsko opremo "firmware").

Različice izvedb mikrokontrolerjev MC6801 se med seboj razlikujejo glede na velikost in tip pomnilnika ROM (EPROM), v velikosti pomnilnika RAM, po hitrosti delovanja (frekvenco urinega signala) in tehnologiji izdelave. Na primer, različica mikrokontrolerja MC68701 vsebuje pomnilnik tipa EPROM. Primeren je torej za manjše serije in za razvojne potrebe. Mikrokontroler MC6803 nima notranjega pomnilnika ROM, vendar je v vsakem drugem pogledu tak kot MC6801. Slika prikazuje blokovno shemo tipičnega mikrokontrolerja MC6801.

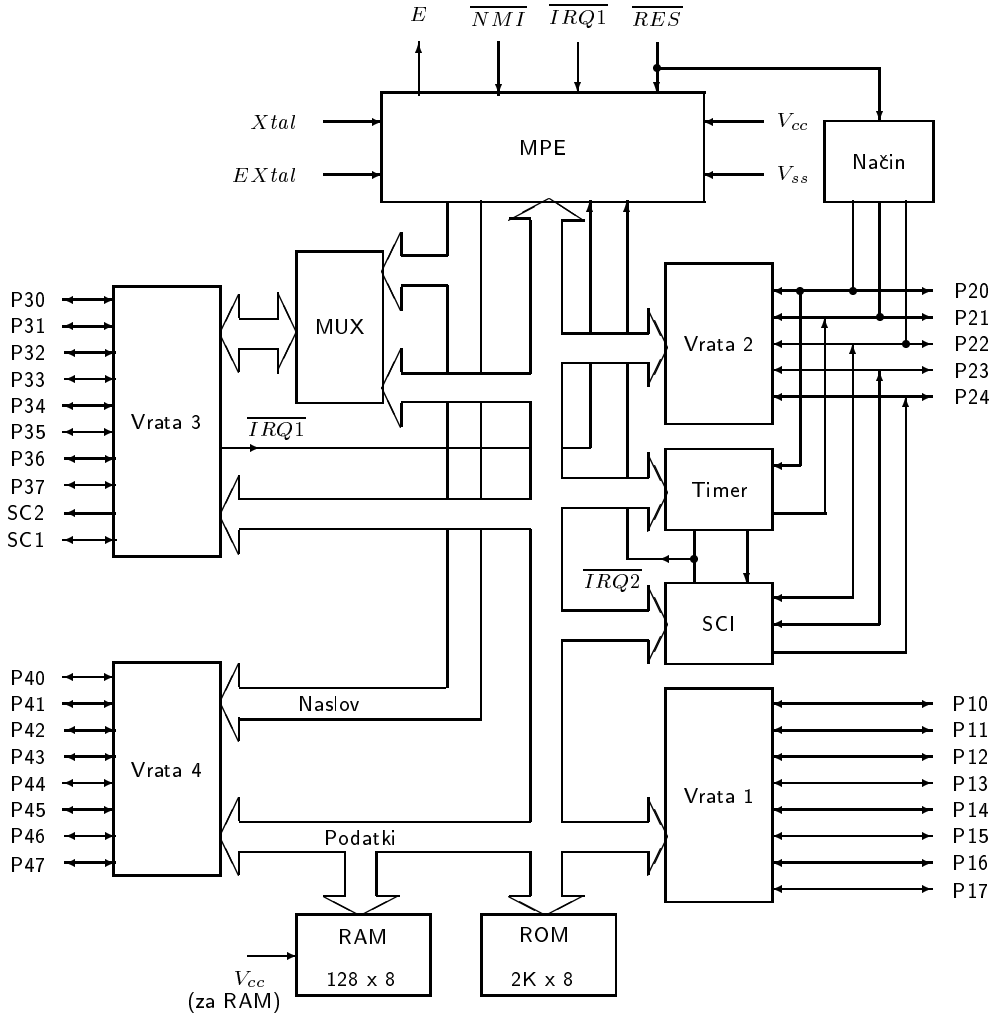
9.1 Načini delovanja

Mikrokontroler MC6801 je zmožen delovati v enem izmed osmih načinov. Način delovanja določi stanje signalov na sponkah P20, P21, P22 ob zagonu (stanje se odbere ob prehodu signala na sponki $\overline{RES}$ v neaktivno stanje). Načine delovanja, imenovane način 0, 1, 2, 3, 4, 5, 6 in 7, lahko razvrstimo v tri temeljne skupine:

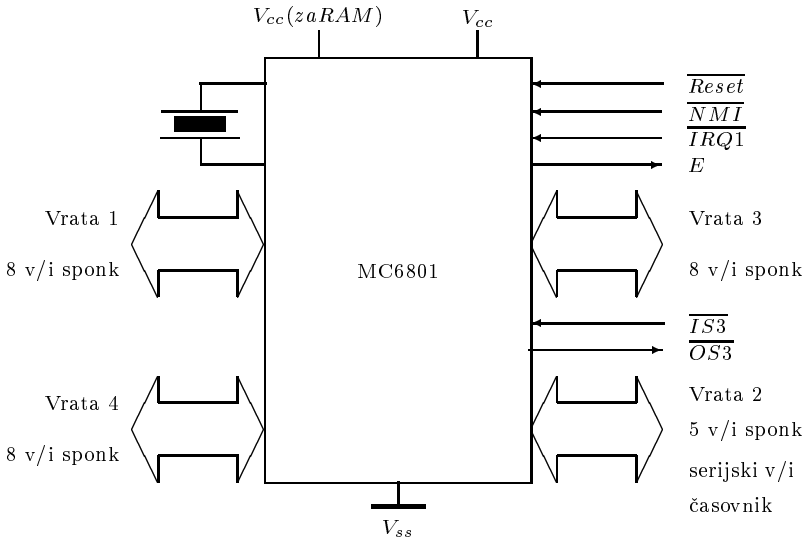
- samostojen način (načina 4 in 7),

- razširjen nemultipleksiran (način 5),
- razširjen multipleksiran način (ostali načini).

Vseh načinov delovanja ne bomo obravnavali. Ogledali si bomo le dva: samostojni način 7 in razširjen multipleksiran način 2 (načina 0 in 4 sta testna načina).

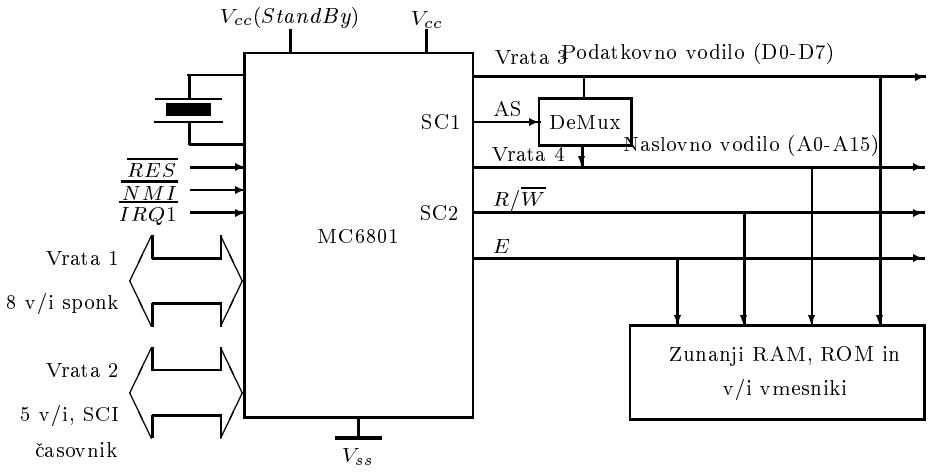


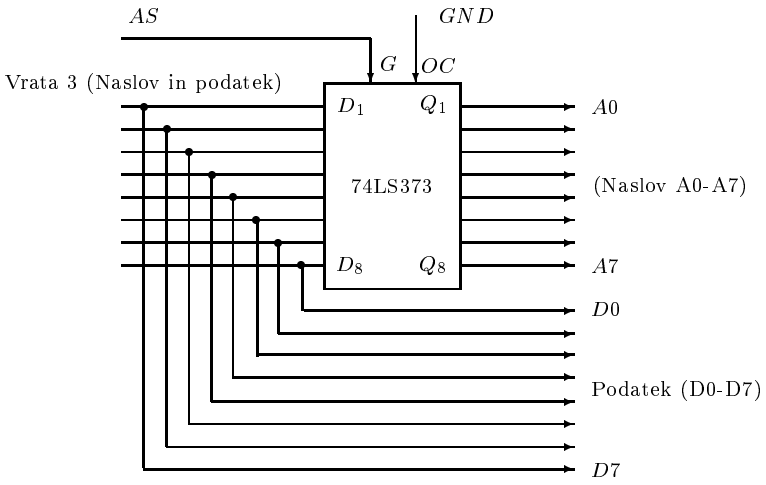
V samosojnem načinu deluje mikrokontroler samostojno, brez dodatnih zunanjih pomnilnih ali perifernih vezij, vse v/i sponke pa so za povezavo mikrokontrolerja z zunanjostjo, Sponki $SC1$ in $SC2$ imata v tem načinu za vrata 3 podobno vlogo kot sponki CA1 in CA2 vmesnika PIA.



Sistem je možno razširiti z zunanjimi vezji (pomnilnikom RAM in EPROM, vmesniki PIA, ACIA, i.t.d.). v tem primeru moramo nekatere v/i sponke žrtvovati.

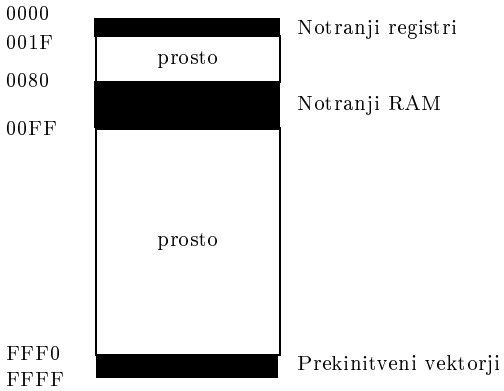
Mikrokontroler v načinu 2 ohranja notranji RAM, a nima notranjega pomnilnika ROM. Način 2 je v bistvu način delovanja mikrokontrolerja MC6803, ki nima notranjega pomnilnika ROM, zato lahko deluje samo v razširjenem načinu. Način 2 bi torej lahko imenovali tudi kar MC6803 način. Vrata 1 dajejo na voljo osem vhodnih ali izhodnih podatkovnih sponk v poljubni kombinaciji, podobno kot kot pri vmesniku PIA. Sponke vrat 2 običajno služijo (poleg določitve načina delovanja ob zagonu) za asinhrono serijsko komunikacijo (vmesnik SCI) in časovnik, tri sponke za SCI in dve za časovnik. Vrata 3 in vrata 4 so za razširitev sistema: vrata 4 služijo za naslovne signale A_8 do A_{15}, na vrata 3 pa so časovno multipleksirani podatkovni signali $D_0 - D_7$ s spodnjimi osmimi naslovnimi signali $A_0 - A_7$. Za demultipleksiranje podatka in naslova služi signal AS (sponka SC1), ki je aktiven, ko je na sponkah prisoten naslov. Tipično vezje (po priporočilu proizvajalca) za demultipleksiranje naslova in podatka prikazuje slika.





9.2 Naslovno področje mikrokontrolerja

Zasedenost naslovnega področja mikrokontrolerja je odvisna od načina delovanja, vendar je v vseh načinih enak položaj notranjih registrov perifernih vezij, pomnilnika RAM (v načinih, ko je omogočen) in prekinitvenih vektorjev.



Notranji registri vrat 1,2,3 in 4, registri časovnega števca in registri serijskega vmesnika SCI se nahajajo od naslova nič naprej, kot prikazuje spodnja tabela. Smerne registre se da le vpisati, berejo pa se kot vrednost $FF. Z vpisom v smerne registre vrat določimo smer perifernih sponk, podobno kot pri vmesniku PIA, le da si smerni registri ne delijo naslova s podatkovnim registrom, vrata 1, 2 in 4 kontrolnega registra nimajo, za vrata 3 pa pride kontrolni/statusni register v poštev samo v samostojnem načinu delovanja mikrokontrolerja.

	Register	Naslov
DDR1	Smerni register (vrata 1)	00
DDR2	Smerni register (vrata 2)	01
DR1	Podatkovni register (vrata 1)	02
DR2	Podatkovni register (vrata 2)	03
DDR3	Smerni register (vrata 3)	04
DDR4	Smerni register (vrata 4)	05
DR3	Podatkovni register (vrata 3)	06
DR4	Podatkovni register (vrata 4)	07
TCSR	Kontrolni/statusni register časovnika	08
TCRH	Časovni števec (zgornji del)	09
TCRL	Časovni števec (spodnji del)	0A
OCRH	Primerjalni register (zgornji del)	0B
OCRL	Primerjalni register (spodnji del)	0C
ICRH	Prestrezni register (zgornji del)	0D
ICRL	Prestrezni register (spodnji del)	0E
CSR3	Kontrolni/statusni register (vrata 3)	0F
RMCR	Hitrost odd./spr. in način	10
TRCSR	Kontrolni/statusni register SCI	11
RDR	Sprejemni register	12
TDR	Oddajni register	13
RAMCR	Kontroni register pomnilnika RAM	14
	Rezervirano	15-1F

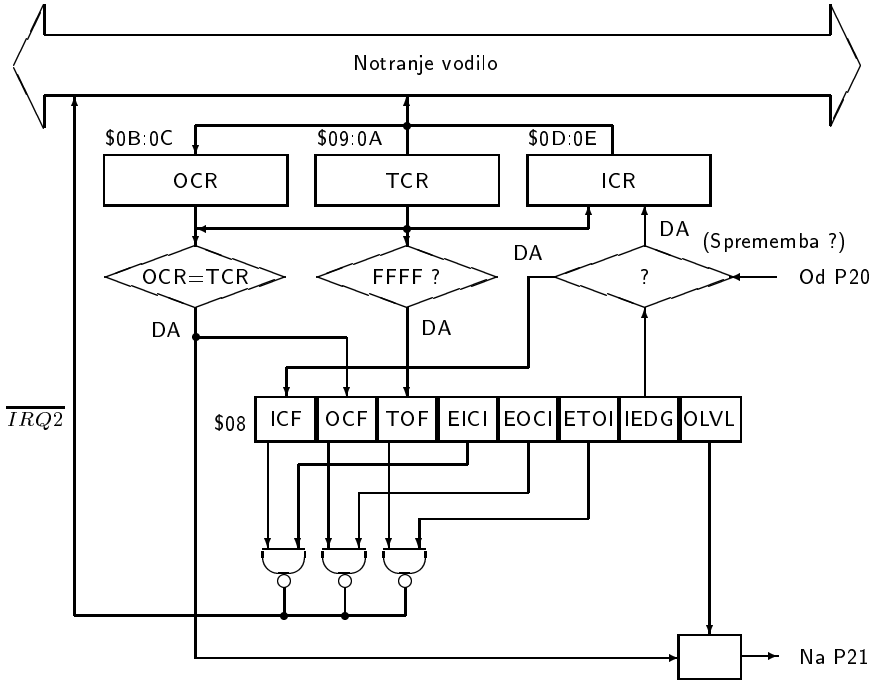
9.3 Časovni števec (časovnik)

Nekaj novosti in zanimivosti za našo obravnavo prinašata časovni števec in serijski komunikacijski vmesnik. Funkcionalno shemo časovnika skupaj s tremi šestnajstbitnimi naslovljivimi registri prikazuje skica. Časovni števec prosto teče, vsebina števnega registra TCR se (krožno) povečuje za ena s signalom E. Ko doseže vrednost $FFFF se postavi bit TOF (Timer OverFlow Flag) in tudi zahteva za prekinitev, če je omogočena z bitom ETOI (Enable Timer OverFlow Interrupt) v kontrolno/statusnem registru (TCSR), štetje pa začne ponovno z nič. Bit TOF se briše z branjem registrov TCSR in TCR (zaporedoma). Vsebina registra TCR se z $\overline{RESET}$ postavi na nič in se je z operacijo vpisa ne da spremeniti, z eno izjemo: vsebina registra se z vpisom (ne glede na vpisano vrednost) postavi na $FFF8, vendar lahko to zmoti obratovanje serijskega vmesnika.

Register OCR (Output Compare Register) koristi za generiranje pravokotnega signala na sponki P21 ali za poljubno časovno kontrolo (na primer otipavanje signalov v enakomernih časovnih presledkih). Ko se vsebina registra OCR ujema z vsebino registra TCR, se vsebina bita OLVL prenese na izhodno sponko P21, postavi se bit OCF in zahteva za prekinitev, če je omogočena z bitom EOCI (Enable Output Compare Interrupt). Bit OCF se briše (če je postavljen) z branjem registra TCSR in vpisom v register OCR.

Register ICR (Input Capture register) se da samo brati. Vanj se prenese vsebina števnega

registra TCR ob aktivni spremembi signala na sponki P20. To postavi bit ICF (Input Capture Flag) in zahtevo za prekinitev, če je omogočena z bitom EICI (Enable Input Capture Interrupt). Bit ICF se briše z branjem registrov TCSR in ICR. Aktivno spremembo (prva ali zadnja fronta signala) določa stanje bita IEDG (Input Edge) in sicer IEDG = 0 za spremebo signala z visokega na nizek nivo.



Pomen posameznih bitov registra TCSR je razviden iz spodnje preglednice.

ICF	(Input Capture Flag)	se postavi ob aktivnem prehodu na sponki P20, prenos TCR → ICR
OCF	(Output Compare Flag)	se postavi ko OCR = TCR
TOF	(Timer OverFlov Flag)	se postavi ko TCR = $FFFF
EICI	(Enable Input Capture Interrupt)	omogoči prekinitev prestreznega načina
EOCI	(Enable Output Compare Interrupt)	omogoči prekinitev primerjalnega načina
ETOI	(Enable Timer OverFlow Interrupt)	omogoči prekinitev, ko se TCR izteče
IEDG	(Input Edge)	za IEDG=0 sledi prenos TCR v ICR ob prehodu signala na P20 navzdol
OLVL	(Output Level)	Nivo na P21, OLVL = 0 za nizek nivo

9.4 Primer programiranja časovnega števca

Poglejmo, kako bi s časovnim števcem generirali pravokotni signal, na sponki P21 mikrokontrolerja, ki vsakih $T = 10$ milisekund spremeni stanje. To dosežemo, če vsakih 10 ms spremenimo stanje bita OLVL iz nič v ena in obratno. V ta namen bomo časovnik pripravili tako, da bo vsakih 10 ms postavil zahtevo za prekinitev, ter sestavili strežni program, ki bo izmenoma postavljal bit OLVL v stanje nič in ena.

Časovni števec je možno pripraviti tako, da povzroči prekinitev vedno, kadar se vsebina primerjalnega registra ujema z vsebino prostotekočega števca. Z zvečanjem vsebine

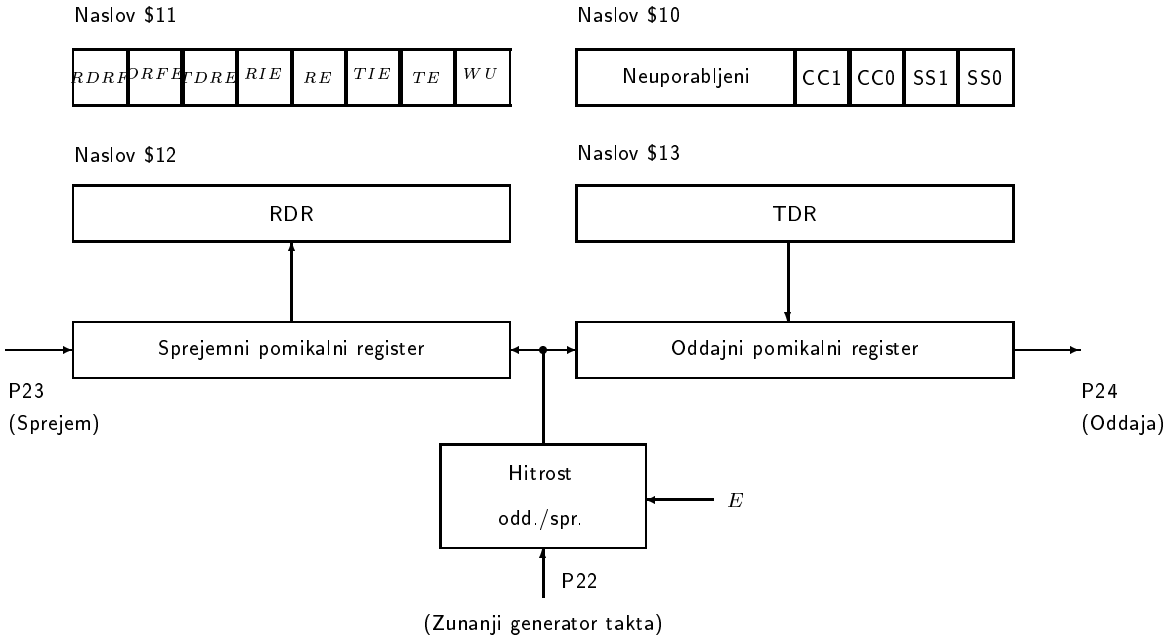
primerjalnega registra za N po vsaki prekinitvi bomo dosegli da se bodo prekinitve vrstile v časovnih presledkih T sekund. Naj bo frekvenca signala E enaka $f_E = 618.\ kHz$. Torej se časovni števec poveča za ena vsakih $T_E = 1/f_E$ sekund, za N pa vsakih $N \times T_E$ sekund. V času $T = 0.010$ sekunde se poveča za

$$N = \frac{T}{T_E} = T \times f_E = 0.01 \times 618000 = 6180$$

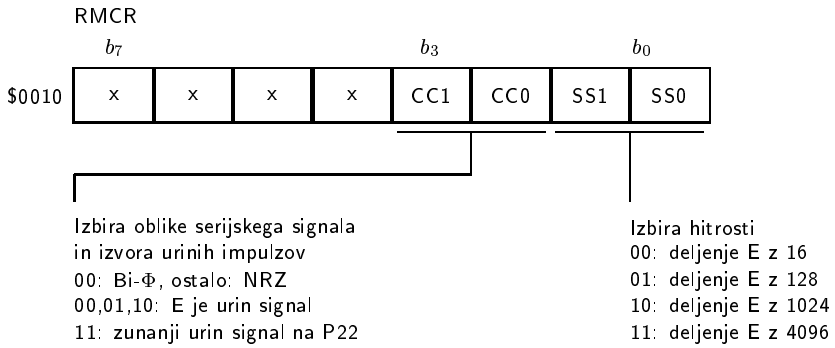
Torej bomo vsebini primerjalni registra prišteli vrednost N vsakič, ko bo prišlo do prekinitve.

9.5 Serijski komunikacijski vmesnik SCI

Serijski komunikacijski vmesnik SCI (Serial Communication Interface) mikrokontrolerja MC6801 je po delovanju in načinu uporabe v marsičem podoben asinhronemu serijskemu vmesniku ACIA MC6850, ki smo ga že obravnavali. Blokovno shemo vmesnika SCI prikazuje slika. Vmesnik med drugim obsega štiri programsko dostopne registre: sprejemni register (RDR), oddajni register (TDR), kontrolno/statusni register TRCSR (Transmit/Receive Control Status register) in register za izbiro hitrosti in načina delovanja RMCR (Rate and Mode Control register). Vsak od teh štirih registrov ima v naslovnem področju procesorja svoj naslov (glej tabelo).



Delovanje SCI se da programirati z vpisom ustreznega vzorca bitov v registra RMCR in TRCSR, vendar je okvir asinhronega serijskega podatka vedno enak: začetni bit, osem podatkovnih bitov brez parnostnega bita in en končni bit. Bit RDRF (Receive Data Register Full) se postavi, ko se vsebina sprejemnega pomikalnega registra prenese v sprejemni podatkovni register RDR, skratka, ko je sprejet nov podatek. Briše se z branjem registra TCSR in RDR. V primeru napake pri sprejemu (prehitevanje podatkov ali napaka sinhronizacije) se postavi bit ORFE (Overrun or Framing Error). Bit se briše z branjem registrov TRCSR in RDR.



Bit TDRE (Transmit Data Register Empty) se postavi, ko se vsebina oddajnega podatkovnega registra prenese v oddajni pomikalni register, torej ko je možna oddaja naslednjega podatka. Briše z branjem registra TRCSR in z vpisom (oddajo) naslednjega podatka v TDR.

Bita TIE (Transmiter Interrupt Enable) in RIE (Reciever Interrupt Enable) omogočata zahtevo za prekinitev $\overline{IRQ2}$ s strani oddajnika in/ali sprejemnika, kot povesta tudi njuni imeni.

Bit TE (transmit Enable) v stanju ena omogoči oddajnik, sponka P24 s tem postane izhodna (oodajna) sponka in taka ostane do ponovnega zagona s signalom $\overline{RESET}$, tudi če se bit TE vrne v stanje nič. Ko se postavi bit TE, oddajnik odda devet zaporednih enic. Bit RE (Receive Enable) ima podobno funkcijo za sprejemnik. Sponka P23 postane vhodna in taka ostane do ponovnega zagona, ne glede na stanje bita RE.

Register RMCR določa hitrost oddajanja in sprejemanja, obliko serijskega signala in izbiro vira pravokotnih impulzov, ki krmilijo generator takta. Z bitoma CC1 in CC0 (Clock Control and Format Select) lahko izbiramo med NRZ (Non-Return-Zero) obliko signala (visok nivo za enico in nizek za ničlo), ki smo je najbolj vajeni (CC1:CC0 = 01,10,11) in bifazno obliko (CC1:CC0 = 00), pri čemer pomeni enico sprememba serijskega signala v sredini trajanja bita, in ničlo nespremenjen nivo signala cel čas oddaje bita). Kombinacija CC1:CC0 = 11 izbere za krmiljenje generatorja takta zunanji signal na sponki P22. Ta je osemkrat višje frekvence od želene hitrosti oddajanja/sprejemanja. Vse ostale kombinacije izberejo za krmiljenje generatorja takta notranji signal (signal E), pri kombinaciji CC1:CC0 = 10 pa se generatorjev takt pojavi tudi na sponki P22.

Kadar generator takta krmili signal E, določa stanje bitov SS1 in SS0 (Speed Select) hitrost oddaje/sprejema. Za stanje bitov SS1:SS0 = 00, 01, 10, 11 se frekvenca signala E deli s 16, 128, 1024 ali 4096. Na primer, pri frekvenci kristala 2.4567 MHz je frekvenca signala E enaka 614.4 KHz in pri stanju bitov SS1:SS0 = 01 dosežemo 128-krat nižjo hitrost oddajanja/sprejemanja, torej 4800 bitov na sekundo.

Zanima lastnost serijskega vmesnika je zmožnost avtomatičnega 'bujenja', ki jo omogočimo z bitom WU (Wake-Up). Ta lastnost pride prav pri komunikaciji večjega števila mikrokontrolerjev MC6801, ki so povezani na skupno serijsko vodilo. Tipičen komunikacijski protokol predvideva naslavljanje (pozivanje) posameznih mikrokontrolerjev z enoznačnim naslovom v začetku sporočila - daljšega zaporedja podatkov. V primeru, da mikrokrmilnik ni pozvan, bo sporočilo enostavno prezrl. Zmožnost bujenja razbremeni mikrokrmilnik stalnega preverjanja stanja na sprejemni liniji.

9.6 Prekinitveni vektorji

MC6801 ima, tako kot MC6800, sponko $\overline{NMI}$, na kateri zahteve za prekinitev ni mogoče preprečiti, in dvoje prekinitvenih signalov ($\overline{IRQ1}$ in $\overline{IRQ2}$, ki ju je možno preprečiti programsko z bitom I v registru CCR:

- $\overline{IRQ1}$ ima prednost pred $\overline{IRQ2}$, streže zahtevi za prekinitev vrat 3 ($\overline{IS3}$) in zahtevam zunanjih naprav (na sponki $\overline{IRQ1}$),
- $\overline{IRQ2}$ streže zahtevam iz časovnika in serijskega vmesnika (SCI) ter ni dostopen od zunaj. Ustrezajo mu štirje prekinitveni vektorji (3 za časovnik) Zahteva za prekinitev iz časovnika ima prednost pred zahtevo iz serijskega vmesnika.

Zgornji del	Spodnji del	Prekinitev
FFFE	FFFF	$\overline{RESET}$
FFFC	FFFD	$\overline{NMI}$
FFFA	FFFB	SWI
FFF8	FFF9	$\overline{IRQ1}$ ali $\overline{IS3}$
FFF6	FFF7	ICF (Input Capture)
FFF4	FFF5	OCF (Output Compare)
FFF2	FFF3	TOF (Timer Overflow)
FFF0	FFF1	SCI (Serial Comm. Interface)