Univerza v Ljubljani
Fakulteta za elektrotehniko

Danjel Vončina

DIGITALNO PROCESIRANJE V MEHATRONIKI - 1

1. del

Zapiski predavanj

Ljubljana, 2013

1. Matematične operacije v dvojiškem številskem sistemu

V dvojiškem številskem sistemu izvajamo matematične operacije po enakih zakonitostih kot v desetiškem sistemu. Zaradi lažje izvedbe digitalnih vezij pa nekatere binarne računske operacije izvajamo na drug način kot smo vajeni v desetiškem sistemu.
Računanje v dvojiškem sistemu je v principu zelo enostavno, saj imamo opravka s samo dvema ciframa. Osnovna pravila seštevanja si bomo ogledali na praktičnem zgledu. Vse ostale operacije, kot so npr. odštevanje, množenje in deljenje pa se pri aritmetični enoti računalnika izvedejo preko operacije seštevanja.

Za seštevanje veljajo naslednja pravila:

- posamezne številke se seštevajo kot v decimalnem sistemu:
  0 + 0 = 0
  1 + 0 = 1
  1 + 1 = 0 (in prenos 1 na naslednje mesto)
  1 + 1 + 1 = 1 (in prenos 1 na naslednje mesto)

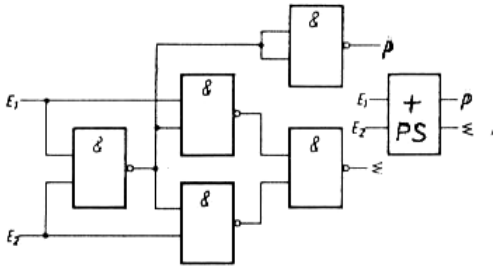
Primeri seštevanja binarnih števil:

	011 (3)		1001 (9)		11.011 (3,375)
+	110 (6)	+	1111 (15)	+	10.110 (2,750)
=	1001 (9)	=	11000 (24)	=	110,001 (6,125)

1.1 Enostavno seštevalno vezje

Aritmetična enota nekega mikroprocesorja je seštevalno vezje, saj vse računske operacije v splošnem prevedemo na seštevanje. Poglejmo si, kako sestavimo takšno enoto s pomočjo logičnih funkcij.

Najbolj enostavno binarno seštevalno vezje je polseštevalno vezje na sliki 1.1. To vezje je z izjemo inverterja pred izhodom P (prenos) identično logični antivalenci oz. »EKSKLUZIVNI ALI«.

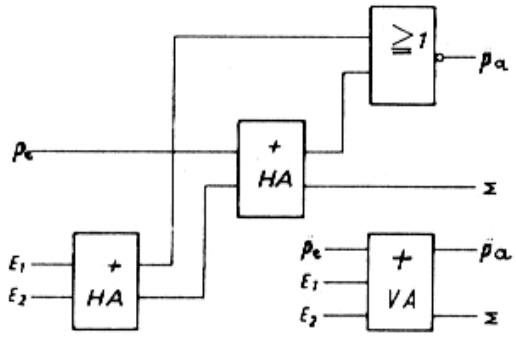


Slika 1.1: Polseštevalno vezje

Tabela 1.1 Izjavnostna tabela polseštevalnega vezja

E_2	E_1	S	P
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Vsota S obeh vhodov E_1 in E_2 sledi pravilu seštevanja binarnih števil. Prenos se pojavi, če seštevamo 1+1. Polseštevalnik ni uporaben za večmestno seštevanje. To reši kaskadna vezava dveh ali več polseštevalnih vezij, ki je prikazana na sliki 1.2.



Slika 1.2: Kaskadna vezava polseštevalnih vezij

Tabela 1.2: Izjavnostna tabela pri kaskadni vezavi dveh polseštevalnih vezij

P_e	E_1	E_2	P_d	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Takšna zaporedna vezava dveh polseštevalnih vezij je že uporabna za seštevanje dveh enomestnih binarnih števil. Dokler je na vhodu P_e vrednost nič, kaže izhod vsoto dveh enomestnih vhodov E_1 in E_2. Če pa je na P_e vrednost 1, je na izhodu vsota treh enomestnih binarnih števil.

1.2 Odštevanje binarnih števil

V večini mikroprocesorskih enot se odštevanje binarnih števil izvaja preko operacije seštevanja z uporabo drugega komplementa. Tako se npr. 7 – 4 = 3 pretvori v 7 + (-4) = 3.
Za osvežitev spomina si najprej poglejmo prvi komplement nekega števila. Prvi komplement števila je vrednost, ki jo prištejemo številu, da dobimo največjo možno vrednost enakomestnega števila.
Npr. v decimalnem sistemu je prvi komplement števil A = 7, B = 53, C = 128:

$$\overline{A} = 9 - 7 = 2$$
$$\overline{B} = 99 - 53 = 46$$
$$\overline{C} = 999 - 128 = 871$$

Pri binarnih številih dobimo prvi komplement tako, da zamenjamo enke z ničlami in obratno.

$$A = 111; \qquad \overline{A} = 000$$
$$B = 11; \qquad \overline{B} = 00$$
$$C = 10101; \qquad \overline{C} = 01010$$

Če seštejemo $A + \overline{A}$, dobimo:

$$\begin{aligned} & 101011 \ (A) \\ + \ & 010100 \ (\overline{A}) \\ = \ & 111111 \ (A + \overline{A}) \end{aligned}$$

Če rezultatu prištejemo ena, dobimo rezultat 0 seveda, če spregledamo prenos. Ta prenos se pojavi na bitnem mestu n+1, ki pa pri n-bitnem procesorju odpade. Za n-bitov je iznos 0. To pa pomeni:

$$A + (\overline{A} + 1) = 0$$
$$(\overline{A} + 1) = -A$$

To je drugi komplement števila A, oziroma njegova negativna vrednost. Poglejmo si sedaj primer odštevanja 7 – 3 = 4 s pomočjo štirimestnega binarnega prikaza. To odštevanje bomo prevedli na seštevanje 7 + (-3) = 4. Pri tem moramo paziti, da binarne vrednosti prikažemo v štiribitnem prikazu.

$$A = 0011 \quad (3)$$
$$\overline{A} + 1 = 1100 + 1 = 1101$$

Seštevanje:

$$\begin{aligned} & A = 0111 \\ + \ & 1101 \quad (-3 \text{ tj. drugi komplement števila } 3) \\ \hline = \ & 10100 \quad (4) \end{aligned}$$

Če gledamo od največ vrednega prenosa naprej, je rezultat pravilen.
Pri odštevanju večjega števila od manjšega je rezultat negativen. Poglejmo si primer odštevanja 3 – 7 = -4.

$$\begin{aligned} A \ &= 0011 \quad (3) \\ -B \ &= 1001 \quad (2. \text{komplement števila } 7) \end{aligned}$$

$$C \ = 1100 \ = \ 12 \text{ oz. } (-4 \text{ ali } -0100)$$

Rezultat je v tem primeru **1**100, kar je -4. Če na ta način predstavljena števila primerjamo, vidimo, da imajo pozitivna števila (od 1 do 7) najpomembnejši bit 0, negativna pa 1. Pri takšni oznaki (ta bit imenujemo tudi predznačni bit) lahko računalnik loči pozitivna in negativna števila in jih tako tudi obdeluje.

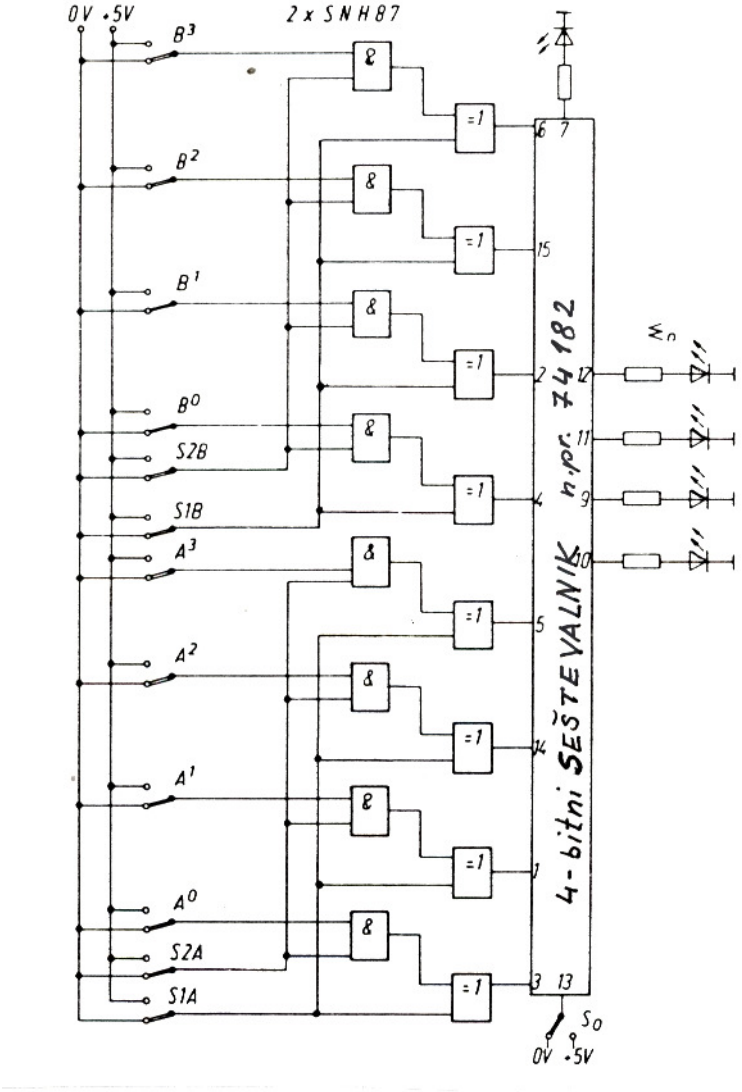
Tabela: 1.3 Delitev pozitivnih in negativnih štirimestnih binarnih števil

A	-A
0000 = 0	(15) 1111 = -1
0001 = 1	(14) 1110 = -2
0010 = 2	(13) 1101 = -3
0011 = 3	(12) 1100 = -4
0100 = 4	(11) 1011 = -5
0101 = 5	(10) 1010 = -6
0110 = 6	(9) 1001 = -7
0111 = 7	(8) 1000 = -8
1000 = 8	

Zgornja tabela nam kaže katera števila lahko predstavimo s štirimestnim binarnim številom ob uporabi drugega komplementa. Imamo možnost zapisa števil od 0 do 15. Bitni vzorci od 0001 do 0111 so prirejeni pozitivnim številom, vzorci od 1000 do 1111 pa negativnim. Izjema je število 8, ki ima enako oznako za pozitivno kot za negativno vrednost.
S petmestnim binarnim številom lahko podamo številski obseg od 0 do31. Bitni vzorci 00001 do 01111 predstavljajo pozitivna števila od 1 do 15, bitni vzorci 10000 do 11111 pa negativna števila od –15 do –1. To velja tudi za druge številske obsege.

1.3 Enostavna aritmetična enota

Ko smo spoznali možnost predstavitve negativnih števil z 2. komplementom, si v tem poglavju oglejmo enostavno realizacijo logičnega vezja, ki omogoča to funkcijo. Na sliki 1.3 je prikazana kompletna aritmetična enota, ki zna seštevati, odštevati, kakor tudi komplementirati ter povečati za 1 (inkrementirati) in pomanjšati za 1 (dekrementirati).



Slika 1.3: Enostavna aritmetična enota

Vezje ima pet krmilnih vhodov S0 do S2B. Stikalo S0 tvori vhod za prenos 4-bitnega seštevalnika C0. V položaju 1 je iznos seštevalnika povečan za 1 (inkrementiran) v položaju 0 pa inkrementiranja ni.

Stikali S1A in S1B predstavljata krmilne vhode za obe skupini po 4 EKSKLUZIVNIH ALI vhodov. S tema vhodoma lahko tvorimo 1. komplement operandov A in B. V položaju 0 sta operanda A in B prenešena preko ekskluzivnih ALI vhodov nespremenjena ($A \veebar 0 = A$ oziroma $B \veebar 0 = B$). V položaju 1 pa sta operanda A oziroma B komplementirana ($A \veebar 1 = \overline{A}$ oziroma $B \veebar 1 = \overline{B}$). Te funkcije poznamo iz izjavnostne tabele funkcije EKSKLUZIVNI ALI.

Stikali S2A in S2B predstavljata IN funkcijo pred vhodi na EKSKLUZIVNI ALI vrata. S tem je omogočena sprostitev oziroma zapora operandov A in B na vhode vrat EKSKLUZIVNI ALI. V položaju 0 imamo zaporo ($A \wedge 0 = 0$ oziroma $B \wedge 0 = 0$), v položaju 1 pa so bitne vrednosti prepuščene na vhode »EKSKLUZIVNI ALI«.

S pomočjo teh pet stikalnih funkcij lahko formiramo $2^5 = 32$ bitnih vzorcev in odgovarjajoče število logičnih funkcij. Nekatere od teh sicer niso zanimive, nekatere pa se ponavljajo. Oglejmo si samo nekatere, ki so zbrane v tabeli 1.4.

Tabela 1.4: Logične funkcije enostavne aritmetične enote s slike 1.3

S2A	S2B	S1A	S1B	S0	N	/
1	0	0	0	0	A	1
0	0	0	0	1	1	2
1	0	1	0	0	$\overline{A}$	3
0	1	0	0	0	B	4
0	0	0	0	0	0	5
1	0	0	0	1	A+1	6
1	0	0	1	0	A-1	7
1	1	0	0	0	A+B	8
1	1	0	1	1	A-B	9
0	0	0	1	0	-1	10

Komentar k tabeli 1.4:

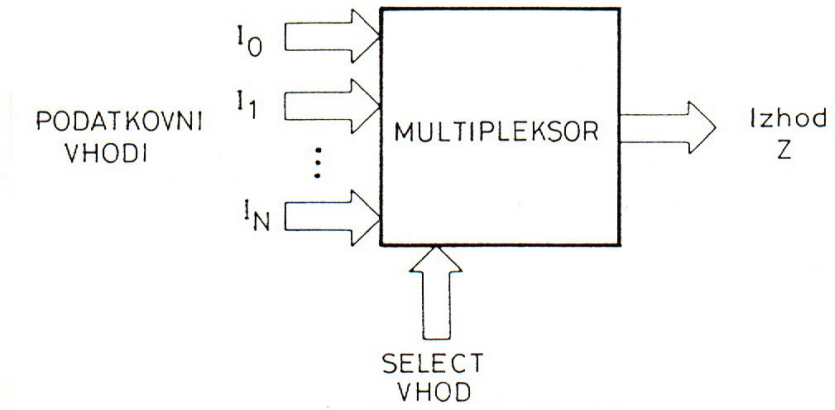
1. Na vhod seštevalnika je s stikalom S2A prepuščen operand A.
2. Nobeden od operandov ni na vhodu seštevalnika. Signal S0 = 1 inkrementira rezultat od 0 na 1.
3. Stikalo S2A prepusti operand A, stikalo S1A pa ga invertira.
4. S stikalom S2B je na izhod prepuščen operand B.
5. Zaradi vseh stikal v položaju 0 je na izhodu 0.
6. S2A spusti operand A, S0 pa ga inkrementira za 1.
7. S2A spusti operand A, S1B invertira vrednost B = 0 v B = -1 (npr. B = 0000, B = 1111 kar pomeni v aritmetiki 2. komplementa –1).
8. S stikali S2A in S2B sta na seštevalniku oba operanda A in B in na izhodu dobimo vsoto A+B.
9. S stikali S2A in S2B sprostimo oba operanda, S1B invertira operand B, S0 mu prišteje 1 (2. komplement) in na izhodu imamo razliko A-B.
10. Stikalo S1B invertira B = 0 v B = -1. Enak rezultat dobimo tudi pri kombinaciji stikal S1A = 1 in S1B = 0.

1.4 Povezava aritmetične enote z logičnimi in z registrskimi enotami

Poleg aritmetične enote nastopajo v mikroprocesorskem sistemu še druge funkcijske enote, ki so potrebne za delovanje celotnega sistema. Mnoge od teh so integrirane že v mikroprocesorskem čipu, ki je v takšni standardni izvedbi že sposoben reševati nezahtevne naloge. Navkljub temu moramo za kompletno razumevanje takšnega sistema razumeti medsebojno delovanje integriranih elementov.

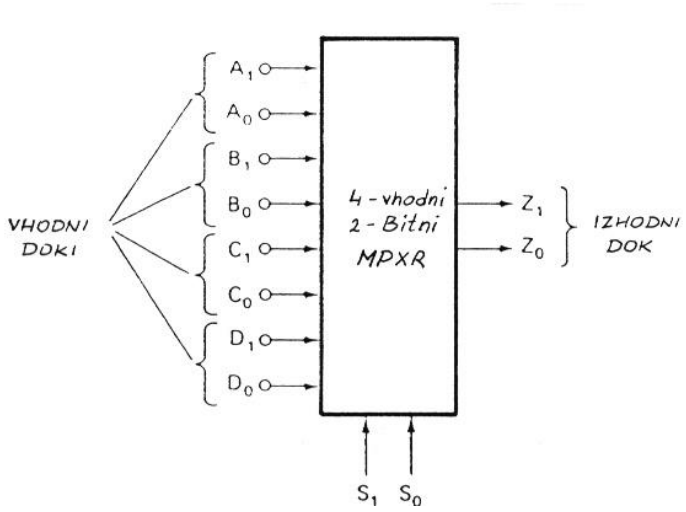
1.4.1 Selektorji podatkov

Razlikujemo dve vrsti selektorjev podatkov in sicer multipleksorje in demultipleksorje. Na sliki 1.4 je prikazan splošen multipleksor. Vidimo, da so tako vhodi kakor tudi izhodi lahko večbitni.



Slika 1.4: Multipleksor

Osnovno delovanje je podobno pri vseh. Na sliki 1.5 imamo primer 4-vhodnega, 2-bitnega multipleksorja. Vsaka skupina vhodov se sestoji iz dveh bitov.



Slika 1.5: Dvobitni štirivhodni multipleksor

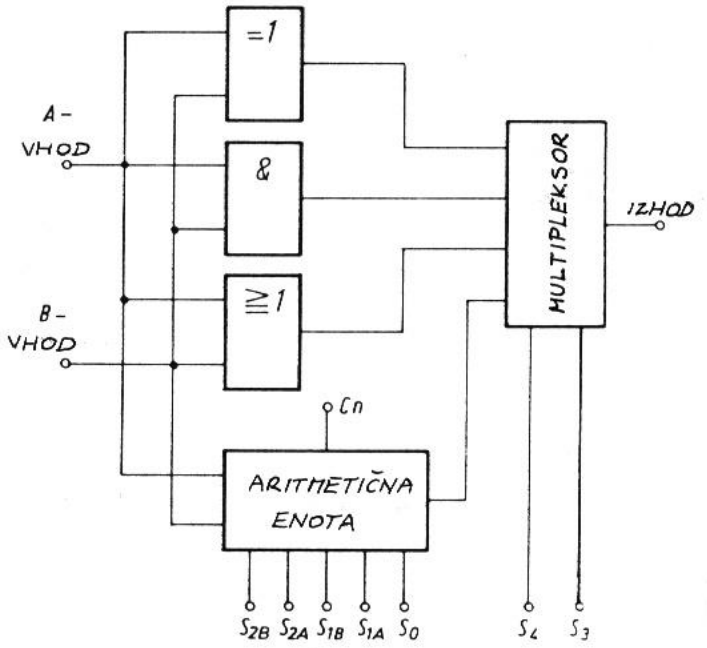
Tabela 1.5 Izjavnostna tabela multipleksorja na sliki 1.5

S_1	S_0	Z_1	Z_0
0	0	A_1	A_0
0	1	B_1	B_0
1	0	C_1	C_0
1	1	D_1	D_0

Imamo štiri take skupine, ki jih imenujemo vhodna vrata. A_1 in A_0 tvorita ena vhodna vrata, B_1 in B_0 pa druga itd. Imamo samo ena izhodna vrata, katerih logični nivoji zavzemajo

vrednosti v tistem trenutku izbranih vhodnih vrat. Le te izbiramo preko vhoda SELECT S_0 in S_1. Delovanje podaja izjavnostna tabela poleg slike multipleksorja. Samo izbrani vhod vpliva na izhod. Drugače povedano, ta multipleksor se obnaša kot multipolarno, multipozicijsko stikalo, katerega stanje je krmiljeno z logičnimi nivoji S_1 in S_0.

Poglejmo primer uporabe multipleksorja, ki razširi funkcijo aritmetične enote v aritmetično-logično enoto (ang. ALU – Arithmetic Logic Unit). To je prikazano na sliki 1.6.



Slika 1.6: Primer aritmetične logične enote (ALE)

S pomočjo 4-bitnega multipleksorja lahko izbiramo med funkcijami:

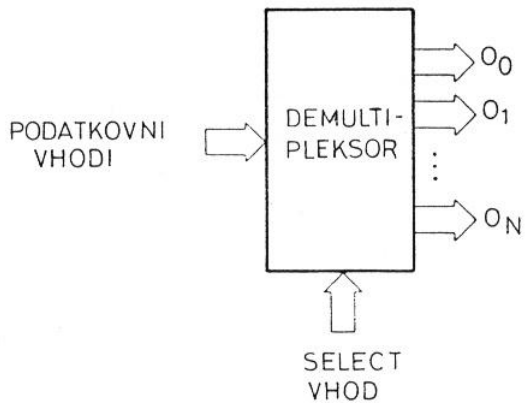
1. Aritmetična enota
2. A V B
3. A Λ B
4. A ∀ B,

ki jih s krmilnima vhodoma S_4 in S_3 posamično lahko peljemo na izhod. V primeru, ko sta S_4 in S_3 v logičnem stanju 1, dobimo na izhodu multipleksorja rezultat, ki ga podaja aritmetična enota. Njeno funkcijo izbiramo s krmilnimi vhodi S_0 do S_{2B}. V spodnji tabeli so zbrane vse praktične funkcije, ki jih zmore enostavna ALU na sliki 1.6. Slika je poenostavljena in je enobitna. Zlahka si razširimo vse vhode na 4 bite in za primer si poglejmo rezultat, ki ga dobimo na izhodu ob prisotnih operandih A = 1101, B = 0011 in ob krmilnih signalih zadnje vrstice v tabeli 10XXXXX.

1101
0011 Rezultat je A, B = 0001
0001

Obstajajo tudi multipleksorji z vmesnim pomnjenjem, ki zagotavljajo status izhodov tudi potem, ko na vhodih ni več signalov.

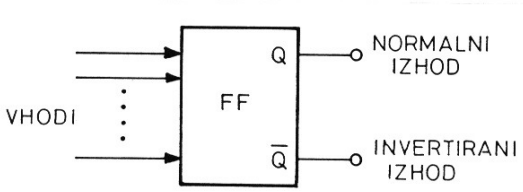
Demultipleksor ima enostavna vhodna vrata in več izhodnih vrat (Slika 1.7). Prenaša podatke oz. vhoda na izbrani izhod, kar določajo izbirni vhodi (select vhodi).



Slika 1.7 Demultipleksor

1.4.2 Flip-flopi in števna vezja

Flip-flopi (FF) so preklopna vezja, ki pri odgovarjajočem krmiljenju lahko prevzemajo 1-bitno informacijo in jo shranijo poljubno dolgo.
FF so logična vezja z dvema izhodoma, ki sta inverzna drug drugemu. Na sliki 1.8 sta ta dva izhoda označena s Q in $\overline{Q}$. Na izhodu $\overline{Q}$ je vedno invertirana vrednost izhoda Q. Če pravimo, da je FF v logičnem stanju 1 (High) ali v logičnem stanju 0 (Low), se to vedno nanaša na izhod Q.



Slika 1.8 Simbol flip-flopa (FF)

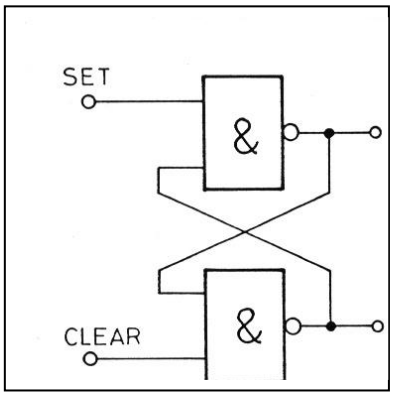
Imamo dve možni delovni stanji FF:

1. $Q = 0,\ \overline{Q} = 1$ in
2. $Q = 1,\ \overline{Q} = 0.$

FF ima enega ali več vhodov, ki lahko povzročijo preklop FF med tema dvema stanjema. Če nek vhodni signal povzroči preklop FF, bo le ta ostal v tem stanju tudi potem, ko ta vhodni signal izgine. To je njegova spominska karakteristika.

1.4.2.1 Osnovno vezje FF

Na sliki 1.9 je prikazano, kako sta uporabljena dva NAND-a križno vezana na osnovni FF (SET/CLEAR FF). Vezje ima dva vhoda, SET in CLEAR. Ta dva vhoda sta normalno oba 1. Če SET pade na 0, gre Q izhod v stanje 1 (in $\overline{Q} = 0$). Tudi če se SET vrne na 1, Q ostane 1 zaradi interne povratne zveze. To se imenuje postavljanje (setiranje) FF. Podobno se zgodi, če CLEAR vhod pade na 0, gre Q izhod na ničlo in tam ostane. To imenujemo brisanje FF.



Slika 1.9: Osnovno vezje FF

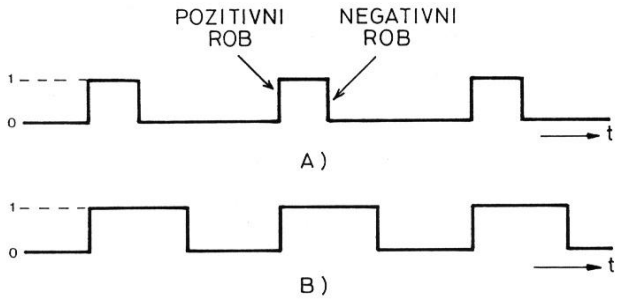
Tabela 1.5: Izjavnostna tabela osnovnega FF

SET	CLEAR	Q
1	1	Ostane nespremenjen
0	1	1
1	0	0
0	0	Dvoumen

Izjavnostna tabela 1.5 podaja delovanje tega FF. Opozorilo: oba vhoda ne smeta biti istočasno v logičnem stanju nič, ker sta potem izhoda dvoumna.

1.4.2.2 Urin signal (CLOCK)

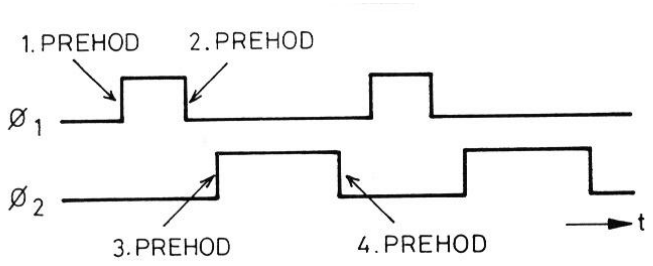
Digitalni sistemi delujejo kot sinhronski sekvenčni sistemi. To pomeni, da se izvaja tista sekvenca, ki je sinhronizirana z urinim ciklom. Ta CLOCK signal ima ponavadi oblike, ki jih kaže slika 1.10. V glavnem je pravokotne oblike in nesimetričen (Slika 1.10a) ali pa simetričen (50/50) kot je prikazan na sliki 1.10b.



Slika 1.10: Urin signal

Clock signal diktira takt delovanja sistema. Posamezni elementi so podrejeni točno določenim stanjem urinega signala in njegovemu prehodu od 1 na 0 in obratno (pozitivni rob, negativni rob).

Večina mikroračunalnikov ima časovni nadzor izveden z dvema ali več takšnimi urinimi signali. Eno takšno običajno kombinacijo kaže slika 1.11. Tam sta uporabljena dva urina signala, ki sta označena s simboloma Φ_1 in Φ_2 (faza 1 in faza 2).



Stanje 1	$\Phi_1 = 0$	$\Phi_2 = 0$
Stanje 2	$\Phi_1 = 1$	$\Phi_2 = 0$
Stanje 3	$\Phi_1 = 0$	$\Phi_2 = 1$

Slika 1.11 Delovanje mikroračunalnika z dvema urinima signaloma

Takšen kompleksen urin signal nudi štiri različne stranice impulzov in tri različna stanja na eno periodo.

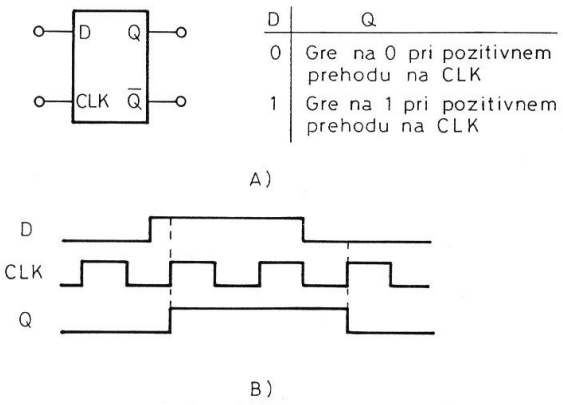
1.4.2.3 Sinhronizirani (dinamični) FLIP-FLOPI

Sinhronizirani FF imajo vsaj dva tipa vhodov: CLOCK vhod (okrajšava CLK) in enega ali več krmilnih vhodov. Krmilni vhodi določijo stanje, ki naj ga zavzame izhod FF po delovanju CLK vhoda. Signal na CLK vhodu je dejanski prožilni signal, ki povzroči, da zavzame FF stanje, ki ustreza krmilnim vhodom.

1.4.2.4 Sinhronizirani D FLIP-FLOP

Slika 1.12a kaže simbol in izjavnostno tabelo za tak sinhroniziran D FF. Ima en sam krmilni vhod D. Deluje tako, da je logični nivo vhoda D prenešen na izhod Q samo pri pozitivnem prehodu na CLK ($0 \rightarrow 1$). D vhod ne vpliva na izhod v vseh drugih časovnih intervalih, kot je to prikazano s časovnimi diagrami signalov na sliki 1.12b.

Negativni prehod signala CLK nima vpliva na izhod FF. Obstajajo tudi sinhronizirani D FF, ki prožijo le ob negativnem prehodu urinega signala (CLK). Ti delujejo sicer popolnoma enako kot prejšnji, v simbolu pa se označujejo s krogom na vhodu CLK.



Slika 1.12: S pozitivnim robom prožen D - FF

1.4.2.5 JK FLIP-FLOP

To je najbolj vsestransko uporabljen FF. Kot vidimo na sliki 1.13 sta uporabljena dva krmilna signala J in K. Ta dva določata, kaj se bo zgodilo s Q izhodom, če pride do pozitivnega prehoda urinega signala.

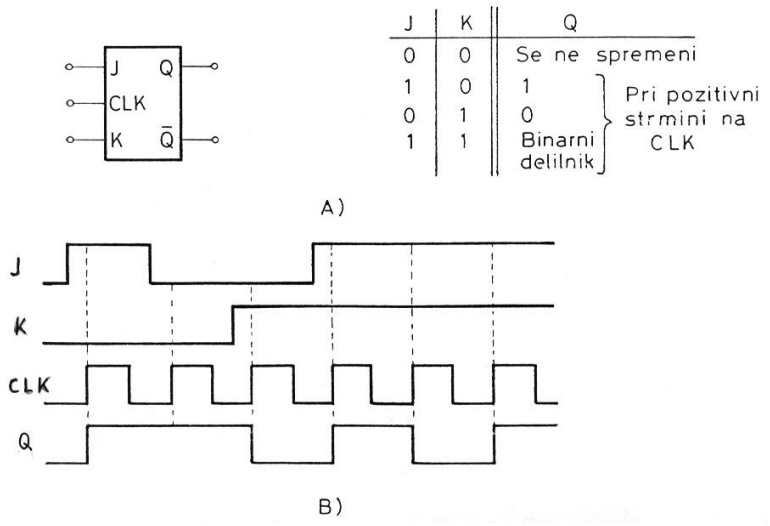
Delovanje:

J = K = 0: V tem stanju pozitivna sprememba CLK ne spremeni stanja na izhodu (glej signala na sl. 1.13.

J = 1, K = 0: Ta pogoj vedno povzroči preklop v stanje Q = 1

J = 0, K = 1: Ta pogoj vedno povzroči Q = 0 ob nastopu CLK.

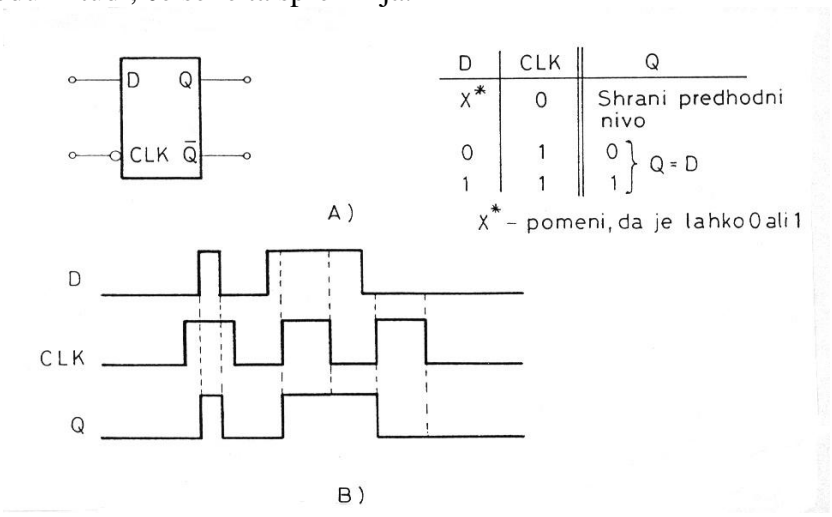
J = 1, K = 1: Če je ta pogoj izpolnjen, potem pri vsakem pozitivnem preklopu ure pride do preklopa Q na inverzno vrednost prejšnjega stanja.



Slika 1.13: Delovanje JK flip-flopa

1.4.2.6 Zapah D (D-type Latch)

Dinamični FF lahko menjajo stanje le pri spremembi CLK. Zapah tipa D je podoben D FF, le da ta lahko spreminja stanje na izhodu med visokim stanjem (HIGH) CLK signala. Slika 1.14 kaže njegov simbol in delovanje. Tako dolgo kot je CLK pozitiven (HIGH), bo izhd Q sledil krmilnemu vhodu D tudi, če se le ta spreminja.



Slika 1.14: Zapah tipa D

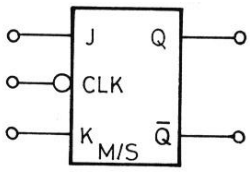
Ko pa je CLK 0 (LOW), Q ohrani (zaskoči) zadnjo vrednost, ki jo je imel in D vhod nima več vpliva na izhod. Krogec ob CLK vhodu pove, da se shrani vrednost vhoda takrat, ko gre CLK v nizko stanje (LOW).

1.4.2.7 JK Master/Slave FF

V nekaterih aplikacijah se lahko zgodi, da krmilni vhodni impulz nastopi približno v istem času kot preklopna strmina CLK vhoda. To lahko privede do nepredvidenega proženja. Da se izognemo temu problemu, je bil razvit JK Master/Slave FF. V splošnem je ta FF sestavljen iz dveh FF (internih – Master in Slave) in vedno proži na negativno strmino CLK vhoda.

Master je vzet zato, da shrani pogoje, ki so na J in K vhodih tik pred nastopom negativne strmine CLK. Če se J in K spremenita istočasno kot CLK, slave reagira tako, kot to narekujejo stari pogoji na vhodih J in K. Na sliki 1.15 je simbol tega FF.

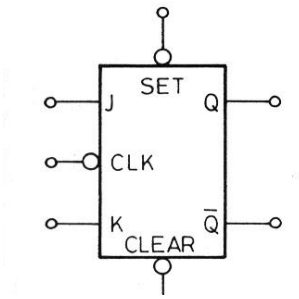
Oznaka M/S ni vedno podana v simbolu, pač pa se v podatkih proizvajalca lako razbere ali gre za običajen JK FF ali pa za Master/Slave.



Slika 1.15: JK Master Slave flip flop

Za sinhronizirane FF, ki smo jih obravnavali, so J, K in D označevali vse krmilne vhode. Ti vhodi so bili imenovani tudi sinhronski vhodi, ker je njihov vpliv na izhod FF sinhroniziran s CLK. Za proženje FF morajo biti sinhronizirani krmilni vhodi vzeti v konjunkciji z urinim signalom.

Mnogi dinamični FF imajo enega ali več vhodov, ki delujejo neodvisno od sinhronizirajočih vhodov in urinega signala. Ti asinhronski vhodi se lahko uporabijo za vstavljanje SET FF (Q = 1) ali za brisanje CLEAR FF (Q = 0) ob vsakem času, ne glede na stanje ostalih vhodov. Drugače rečeno, asinhronski vhodi so torej »nadrejeni vhodi«, ki prevladujejo vse ostale vhode pri postavljanju FF v eno ali v drugo stanje.



SET	CLEAR	ODGOVOR FF
0	0	DVOUMEN
0	1	SET Q = 1
1	0	CLEAR Q=0
1	1	ne vpliva na FF

Slika 1.16: JK FF z asinhronskimi vhodi

Na sliki 1.16 je prikazana slika J-K FF z vhodom SET in CLEAR. Ta dva asinhronska vhoda sta aktivirana pri nivoju 0, kar simbolizira majhen krogec na FF simbolu. Pripadajoča izjavnostna tabela kaže, kako delata ta dva vhoda.

Signal 0 (LOW) na SET vhodu brezpogojno postavi Q v stanje 1. Signal 0 na CLEAR brezpogojno postavi Q na vrednost 0. Istočasno nizko stanje na dveh vhodih je prepovedano stanje, ker to pelje v nedoločeno oz. dvoumno stanje. Če nobeden od obeh vhodov ni LOW (0), potem je FF prost in upošteva vhode J, K in CLK kot je opisano.

Asinhronski vhodi vplivajo torej z enosmernim nivojem (stanjem). To pomeni, da je pri konstantni nič na SET vhodu na izhodu Q = 1 ne glede na to, kaj je na drugih vhodih. Podobno konstantni 0 na CLEAR vhodu pomeni na izhodu Q = 0. Asinhronski vhodi omogočajo držanje FF v posebnem stanju za določen časovni interval. Večkrat so asinhronski vhodi uporabljeni za postavljanje FF v želeno stanje.

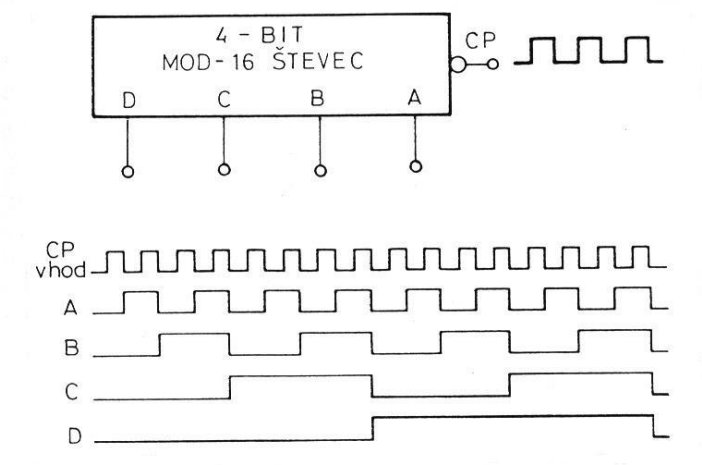
1.4.3 Binarni števci

FF so osnovni sestavni deli števcev. Le ti so izvedeni kot standardna integrirana vezja in jih ni treba več sestavljati iz individualnih FF. Zaradi tega se ne bomo poglabljali v interno delovanje števcev pač pa bomo pogledali le njihove zunanje karakteristike.

1.4.3.1 Delovanje osnovnega števca

Na sliki 1.17 je shematski prikaz 4 bitnega števca. Ta števec vsebuje 4 FF, vsakega za en bit. Njihovi izhodi so označeni z A, B, C in D. Vidna sta dva vhoda; vhod za urin signal (clockpulse - CP) in vhod za RESET. Števec deluje tako, da štirje FF kažejo binarno število, ki je enako številu pulzov, ki se posredujejo na vhod CP. Diagram kaže posamezne sekvence,

ki jim sledijo izhodi FF, ko prihajajo vhodni impulzi. Izhod A je najmanj pomembno mesto (LSB – Least Significant Bit), izhod D pa najpomembnejše mesto binarnega štetja (MSB – Most Significant Bit). Na primer, po petih vhodnih impulzih je izhod DCBA = 0101.



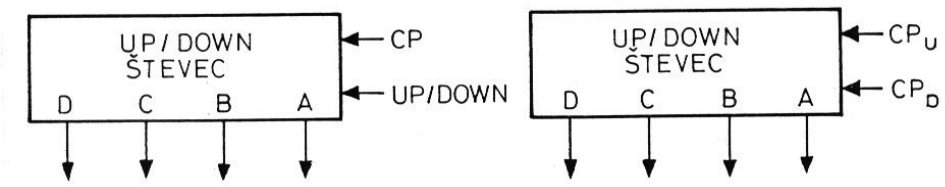
Slika 1.17: Delovanje 4-bitnega števca

Ta štiribitni števec lahko šteje od 0000_2 do 1111_2 tj. vseh 16 možnih stanj. Ko doseže 1111, se ob naslednjem vhodnem impulzu postavijo izhodi FF na vrednost 0000. Štetje se ponovi z naslednjo pulzno sekvenco. V splošnem lahko števec z n FF šteje od 0 do 2^n različnih stanj.

Celoten števni obseg imenujemo tudi COUNTERS MOD-NUMBER ali značilno števno število. Števno sekvenco lahko na kratko prekinemo ob vsakem času s postavitvijo ustreznega logičnega nivoja na RESET vhodu. Na sliki 1.17 visoko stanje na reset brezpogojno resetira vse FF na stanje 0 in jih drži na tem nivoju, dokler se RESET ne vrne na nič. Mnodi IC števci so v notranjosti zgrajeni tako, da ne štejejo skozi kompletno sekvenco 2^{n-1}, ampak do nekega manjšega števila in se potem ponovno postavijo na 0. Npr. ti. dekadni števec, ki šteje od 0000 do 1001 in se potem postavi na 0 ima deset različnih stanj oz. ga imenujemo MOD-10 ali BCD števec saj FF štejejo v desetiški kodi. Nekateri standardni števci so oblikovani tako, da jim lahko od zunaj z zunanjimi vezji določimo števno sekvenco. To so ti. VARIABLE MOD – Counters.

1.4.3.2 Dvosmerni števci

Števci o katerih smo govorili znajo šteti od 0 do nekega maksimalnega števila in se potem postavijo na nič. Obstajajo pa različni števci, ki znajo šteti v dveh smereh in jih imenujemo UP/DOWN ali dvosmerni števci. Slika 1.18 kaže dve osnovni dvosmerni števni vezji.



a) b)

Slika 1.18: Dve izvedbi dvosmernega števca

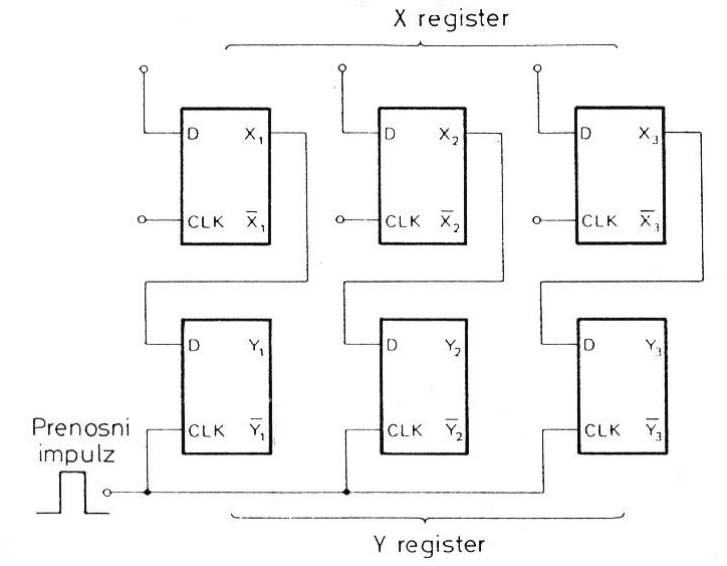
Števec na sliki 1.18a ima enostaven CP vhod, ki je vzet za oba primera: štetje naprej in štetje nazaj. Vhod, označen z UP/DOWN določa smer štetja. En logični nivo na tem vhodu povzroči štetje navzgor od 0000 do 1111, drugi logični nivo na vhodu UP/DOWN pa obrne smer štetja od 1111 do 0000.
Števec na sliki 1.18b nima krmilnega vhoda UP-DOWN, pač pa ima posamične krmilne vhode za štetje navzgor CP_U in za štetje navzdol. Krmilni signal sme biti istočasno le na enem CP vhodu, sicer pride do nesporazuma.

1.4.4 FF registri

Enostaven register je skupina spominskih celic, ki so namenjene shranjevanju binarne informacije. Bolj kompleksni registri lahko modificirajo shranjeno informacijo na več načinov. Skupno vezje vseh registrov je flip-flop.
Registri igrajo zelo pomembno vlogo v mikroprocesorski tehniki. Dejansko je mikroprocesor oz. mikrokrmilnik v poenostavljeni obliki sistem registrov, pri čemer se binarne informacije prenašajo iz enega v drugega.
Oglejmo si prenos informacij iz enega registra v drugega.

1.4.4.1 Paralelni prenos

Na sliki 1.19 imamo dva 3-bitna registra. X register je sestavljen iz FF - X_1, X_2 in X_3; Y register pa sestavljajo FF - Y_1, Y_2 in Y_3.



Slika 1.19: Paralelni prenos podatkov med dvema registroma

Vsak FF je sinhroniziran D FF. Prenosni impulz (TRANSFER) na CLK vhod Y registrov povzroči, da se nivo X_1 prenese v Y_1, X_2 v Y_2 in X_3 v Y_3. Ta prenos vsebine X registra v Y register je paralelni prenos, ker so bili 3 biti prenešeni istočasno. Ker smo imeli opravka s prenosom vseh bitov iz registra v register, bomo za tak prenos uporabili sledečo oznako:

$$[X] \rightarrow [Y]$$

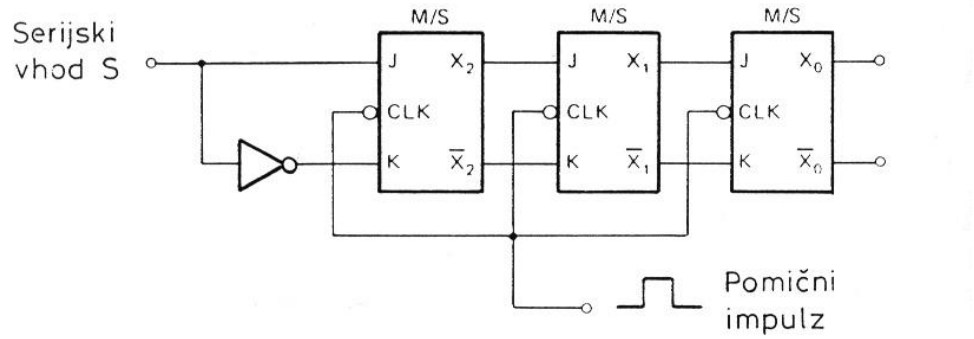
[X] - vsebina X registra
[Y] - vsebina Y registra
→ - smer prenosa

1.4.4.2 Serijski prenos

Pri serijskem prenosu se prenese vsakokrat 1 bit. Predno raziščemo serijski prenos, si oglejmo delovanje pomičnega registra. Slika 1.20 kaže 3-bitni pomični register ob uporabi Master/Slave JK FF-ov. FF so priključeni tako, da je trenutna vsebina vsakega FF ob nastopu negativne strmine pomičnega impulza prenešena v FF na njegovi desni strani. Pri tem se nivo na serijskem vhodu S prenese v FF X_2. Poglejmo primer, ko je vsebina registra X: $X_2 = 1$, $X_1 = 0$, $X_0 = 1$. V skrajšani obliki lahko zapišemo X = 101. Nivo na serijskem vhodu pa naj bo 0. Ob nastopu pomičnega impulza se vsak od teh nivojev prenese na naslednji FF. Torej je po nastopu negativne strmine CLK impulza vsebina registra X = 010. Nivo vhoda S se je prenesel v X_2, predhodni nivo X_2 se je premaknil v X_1 in predhodni nivo X_1 se je premaknil v X_0. Predhodni nivo X_0 se je izgubil.

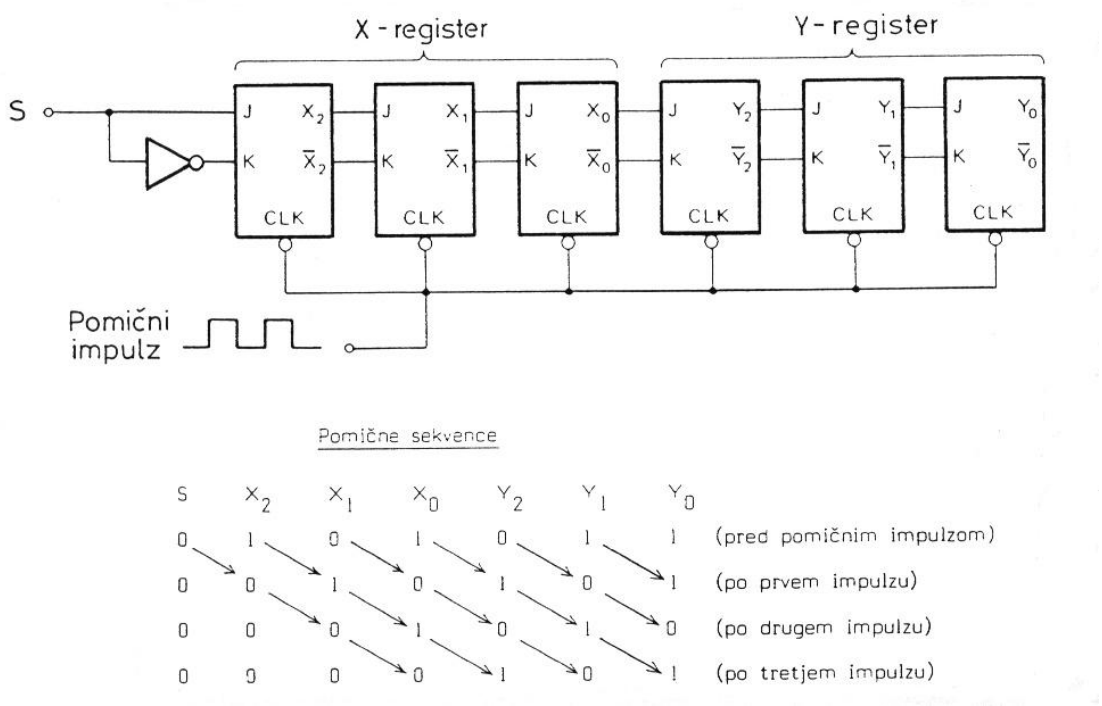
Če se je S spremenil na 1 in če nastopi drugi pomični impulz, je rezultat X = 101. Pri S = 1 bi tretji pomični impulz povzročil rezultat X = 110, itd.

Tudi pomični registri so lahko izvedeni s FF. V obeh primerih morajo biti FF tipa Master/Slave za pravilen prenos. To je predvsem zato, ker se nivoji na krmilnih vhodih (J, K in D) lahko spremenijo v času, ko nastopi negativna strmina pomičnega impulza.



Slika 1.20: Pomični register

Na sliki 1.21 sta prikazana dva 3-bitna pomična serijska registra, ki sta povezana tako, da se vsebina X registra lahko serijsko prenese v Y register. Kompleten prenos se izvrši po treh pomičnih impulzih. Prenosne sekvence so razvidne v diagramu, pri čemer je na začetku S = 0, X = 101, in Y = 011.

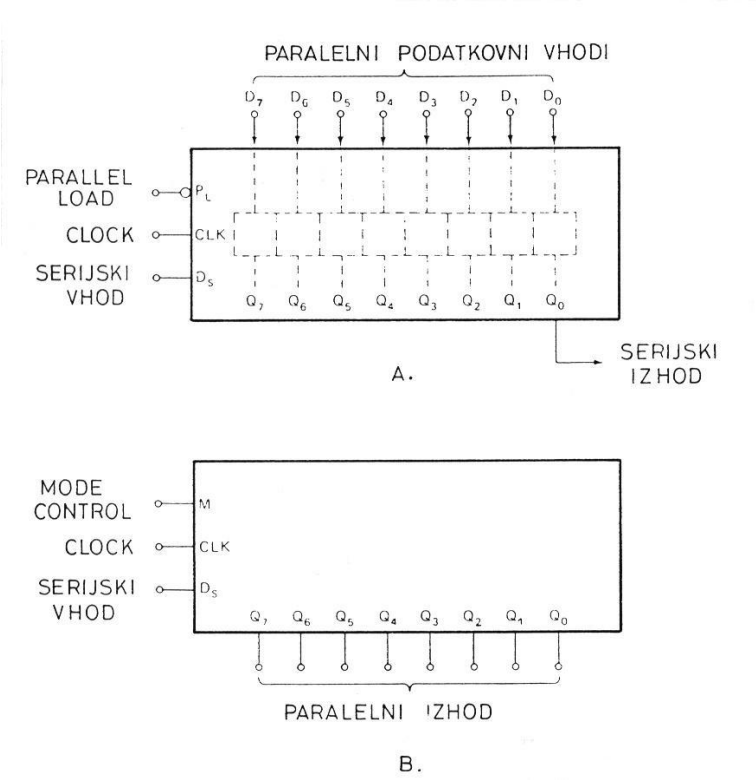


Slika 1.21: Serijski prenos podatka iz enega v drugi register

Serijski prenos zahteva več časa kot paralelni prenos, ker potrebuje za prenos vsakega bita eno časovno periodo CLK. Po drugi strani pa serijski prenos zahteva manj povezav med dvema registroma.

1.4.4.3 Pomični registri

Obstajajo različne izvedbe osnovnih pomičnih registrov. Razlikujejo se po tem, kako podatki vstopajo v register (serijsko, paralelno in kombinacija obeh), kakor tudi po tem, ali so izhodni podatki v serijski ali v paralelni obliki. Prav tako imajo krmilne vhode za določanje smeri pomika (z leve proti desni ali obratno). Za ilustracijo sta na sliki 1.22 prikazana bloka dveh običajnih integriranih pomičnih registrov.



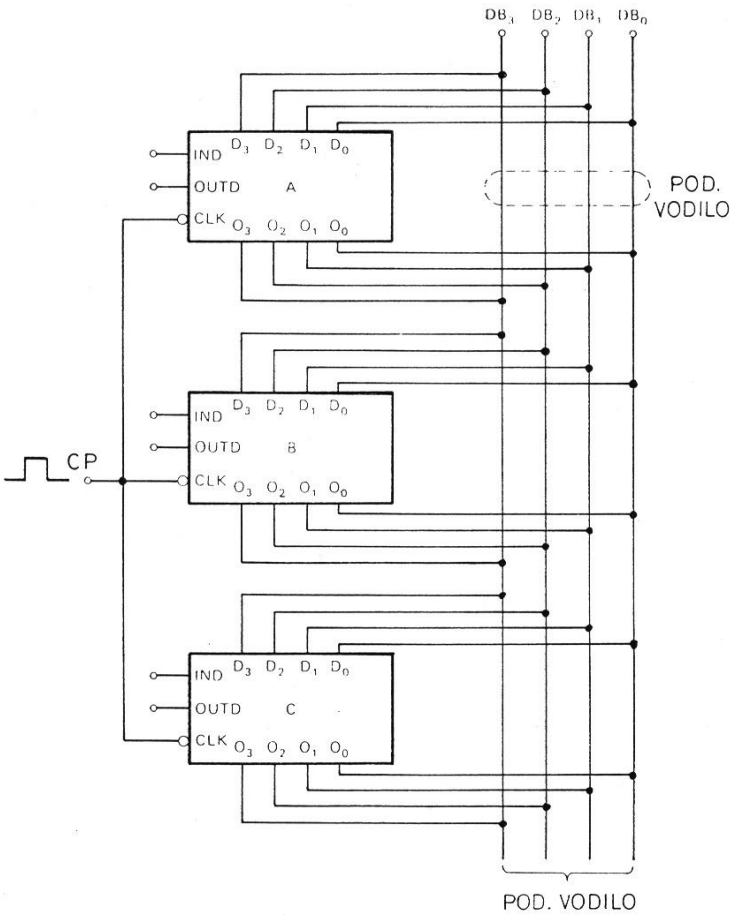
Slika 1.22: Pomični register

IC na sliki 1.22a se imenuje »Parallel-in, Serial-out« pomični register. Sestavljen je iz 8 FF, ki so prirejeni za pomik podatkov od leve proti desni pri prehodu CLK signala od nizkega proti visokemu nivoju. Podatki lahko vstopajo v register po dveh različnih poteh. Serijski podatek lahko vstopa skozi serijski vhod D_S. Ta podatek se ob preklopu CLK premakne v Q_7. Paralelni podatki lahko vstopajo skozi vhode D_0 do D_7 ob aktivnem signalu na paralelnem vhodu »parallel load«. Kadarkoli gre P_L v nizko stanje, so nivoji na D0 do D7 neposredno prenešeni v FF registra. Če je P_L v visokem stanju, so paralelni podatkovni vhodi onesposobljeni.
To integrirano vezje ima samo en FF izhod Q_0. Ta predstavlja skrajno desni bit, kar pomeni, da se podatki pomikajo od leve proti desni. Razlog, zakaj izhodi Q_1 do Q_7 niso dostopni, je omejeno število priključnih nožic (običajno 14 ali 16).

Slika 1.22b kaže pomični register, ki ima serijski vhod in paralelni izhod. Podatki lahko vstopajo v ta register le serijsko skozi D_S pri spremembi CLK z nizkega nivoja na visokega. Vseh osem FF izhodov je dostopnih za zunanjo uporabo po preteku ustreznega števila urinih impulzov (CLK). Krmilni vhod M (mode control) je namenjen določanju smeri pomika. Pri enem logičnem nivoju na M je pomikanje od leve proti desni, pri drugem pa od desne proti levi.

1.4.4.4 Operacija prenosa podatkov iz enega registra v drugega

Na sliki 1.23 imamo tri registre na skupnem podatkovnem vodilu. Poglejmo sedaj prenos vsebine iz registra A v register C.



Slika 1.23: Prenos podatkov med registri po skupnme vodilu

Za izvedbo tega prenosa morajo biti izhodni podatki registra A prenešeni na podatkovna vodila, kar se zgodi, če dobi krmilni vhod $OUTD_A = 1$. Prav tako mora imeti krmilni vhod registra C IND = 1 tako, da lahko sprejema podatke. Vsi ostali D (disable) vhodi morajo biti v stanju 0. Tedaj morajo biti izpolnjeni naslednji pogoji:

$$IND_A = 0, \quad OUTD_A = 1$$
$$IND_B = 0, \quad OUTD_B = 0$$
$$IND_C = 1, \quad OUTD_C = 0$$

S temi nivoji na Disable vhodih bo preklop od 1 na 0 signala CLK povzročil, da se bodo logični nivoji na podatkovnem vodilu (izhodi registra A) prenesli v register C.

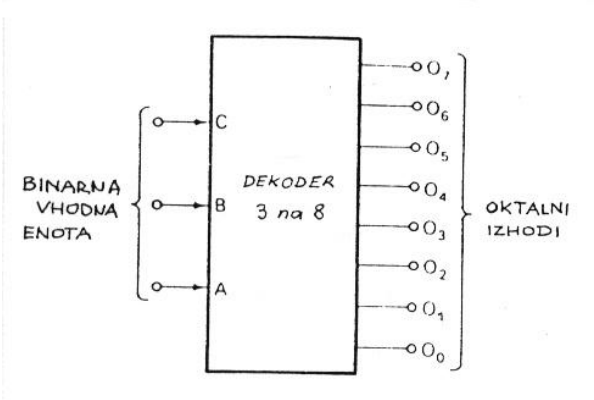
S slike 1.23 je razvidno, da je razširitev registrskega vezja enostavna. Vsak dodatni register pomeni dva dodatna »Disable« vhoda, ki sta potrebna za krmiljenje prenosa podatkov posamičnega registra.

1.4.5 Dekoderji in enkoderji

V digitalnih računalnikih je binarna koda uporabljena za podajanje različnih tipov informacij, kot so: znaki (ASCII koda), numerični podatki, naslovi spominskih besed in krmilni ukazi. Kodna skupina, ki vsebuje N-bitov, ima lahko 2^N različnih kombinacij.

Dekoderji so logična vezja, ki na osnovi N-bitne kode na vhodu generirajo pripadajoč izhodni signal.

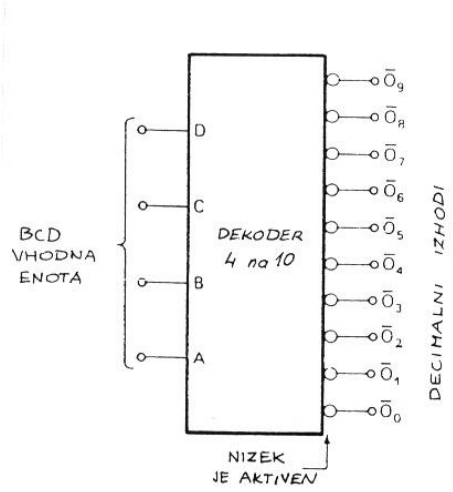
Večina dekoderjev v integrirani izvedbi lahko dekodira 1-, 3- ali 4-bitno vhodno kodo. Takšni primeri so podani na sliki 1.24. Na sliki zgoraj je dekoder 3 na 8 linij. Ima 3-bitni vhod (C,B,A) in $2^3 = 8$ možnih kombinacij. Delovanje je razvidno iz izjavnostne tabele. Na primer, če je vhodna koda CBA = 100 (binarni ekvivalent od 4_{10}), potem je izhod 4 v stanju logične 1, vsi ostali izhodi pa so v stanju logične 0. Tak dekoder imenujemo binarno oktalni dekoder, ker z binarno vhodno kodo izbira 8 izhodov.



Izjavnostna tabela:

C	B	A	Izhodi
0	0	0	$Q_0 = 1$; vsi ostali so 0
0	0	1	$Q_1 = 1$; vsi ostali so 0
0	1	0	$Q_2 = 1$; vsi ostali so 0
0	1	1	$Q_3 = 1$; vsi ostali so 0
1	0	0	$Q_4 = 1$; vsi ostali so 0
1	0	1	$Q_5 = 1$; vsi ostali so 0
1	1	0	$Q_6 = 1$; vsi ostali so 0
1	1	1	$Q_7 = 1$; vsi ostali so 0

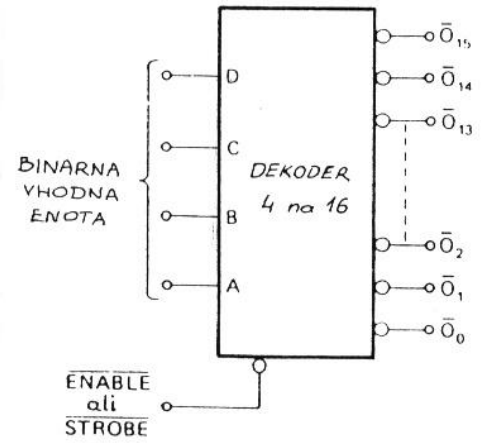
a)



Izjavnostna tabela:

D	C	B	A	Izhodi
0	0	0	0	$Q_0 = 1$; vsi ostali so 0
0	0	0	1	$Q_1 = 1$; vsi ostali so 0
0	0	1	0	$Q_2 = 1$; vsi ostali so 0
0	0	1	1	$Q_3 = 1$; vsi ostali so 0
0	1	0	0	$Q_4 = 1$; vsi ostali so 0
0	1	0	1	$Q_5 = 1$; vsi ostali so 0
0	1	1	0	$Q_6 = 1$; vsi ostali so 0
0	1	1	1	$Q_7 = 1$; vsi ostali so 0
1	0	0	0	$Q_8 = 1$; vsi ostali so 0
1	0	0	1	$Q_9 = 1$; vsi ostali so 0
1	0	1	0	Vsi izhodi v stanju 1
1	0	1	1	Vsi izhodi v stanju 1
1	1	0	0	Vsi izhodi v stanju 1
1	1	0	1	Vsi izhodi v stanju 1
1	1	1	0	Vsi izhodi v stanju 1
1	1	1	1	Vsi izhodi v stanju 1

b)



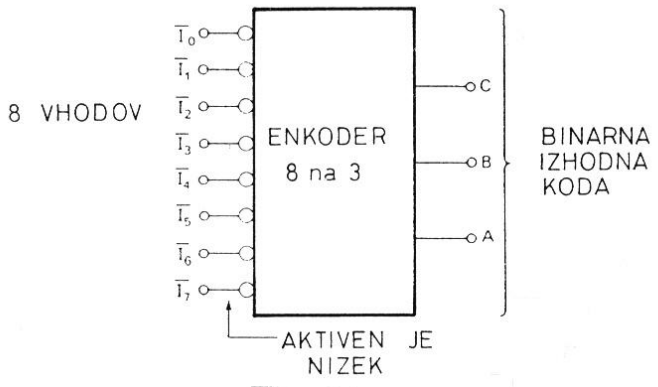
Delovanje:

(1) ENABLE = 0; Deluje kot normalendekoder z aktivnimi izhodi LOW.
(2) ENABLE = 1; Vsi izhodi preklopijo v visoko stanje.

c)

Slika 1.24: Dekodirna vezja

Dekoder na sliki 1.24c vsebuje 16 možnih kombinacij štirih vhodov, ker ima 16 izhodov, ki so aktivni v nizkem stanju. Posebni krmilni vhod (ENABLE – omogočiti), večkrat označen tudi kot STROBE (injicirati) je namenjen krmiljenju delovanja dekoderja. Majhen krogec na vhodu označuje, da je dekoder aktiven v nizkem stanju. Ko je na ENABLE vhodu logična 0, lahko dekoder normalno deluje. Pri logičnem stanju 1 na na vhodu ENABLE so vsi izhodi v visokem stanju, neodvisno od vhodne binarne kode.



Slika 1.25: Enkoder

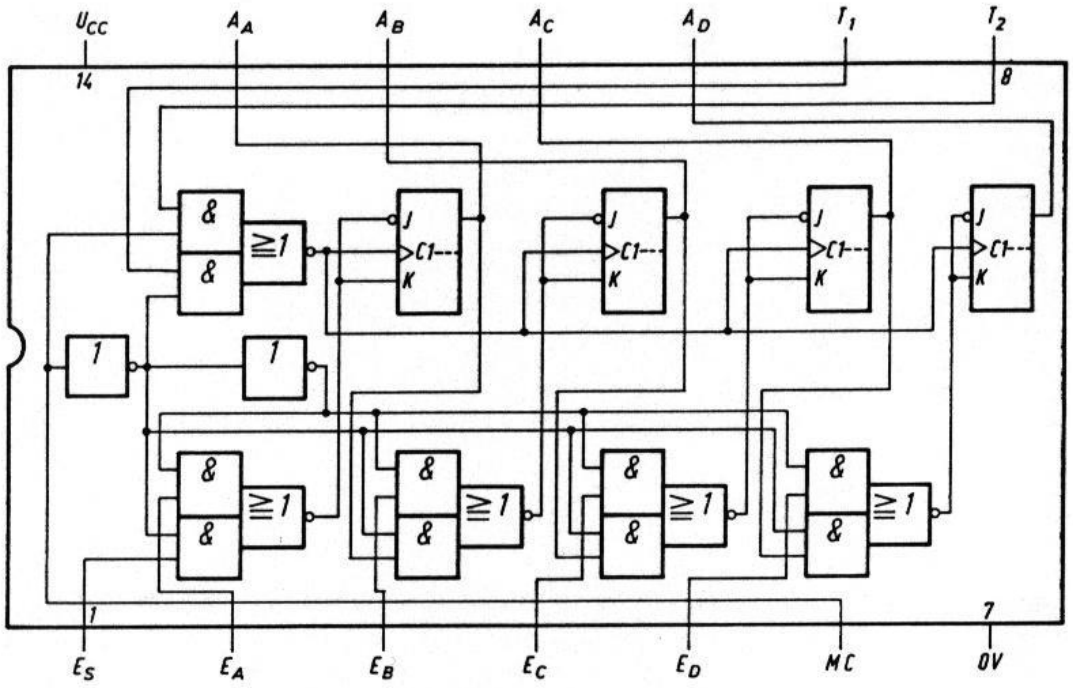
Dekoder glede na vhodno kodo določi izhod. Enkoder deluje ravno obratno; oblikuje binarno izhodno kodo glede na to, kateri vhod je aktiven. Enkoder na sliki 1.25 ima 8 vhodov, ki so aktivni v nizkem stanju. Glede na to, kateri vhod je na nivoju 0, se vzpostavi izhodna binarna koda. Če je npr. $I_3 = 0$, je izhod CBA = 011, kar je binarni ekvivalent 3_{10}. Če je $I_6 = 0$, je izhod CBA = 110.

1.4.6 Pomični register v vlogi akumulatorja za razširitev ALE

Prej predstavljena ALE lahko izvaja le aritmetično logične funkcije. Za hranjenje rezultata in za izvajanje pomikalnih operacij rabimo naslednji gradnik, ki ga imenujemo akumulator. Realiziramo ga s pomočjo pomičnega registra.

Kot bomo kasneje videli, so registri v mikroračunalniku (npr. naslovni register, podatkovni register, indeksni register, register kazalcev sklada, vhodno-izhodni register, instrukcijski register) pravzaprav pomični registri.

Tak 4-bitni pomični register za obe smeri pomika je prikazan na sliki 1.26. Omogoča serijski in paralelni vnos in iznos podatkov.



Slika 1.26: 4-bitni pomični register

Serijski vnos podatkov je možen preko vhoda E_S, če je krmilni vhod MC (Mode Control) v nizkem logičnem stanju in ob prisotnosti urinega signala na vhodu T_1.

Prenos informacije iz E_S na izhode JK-FF se izvrši pri negativni strmini urinega signala na T_1. Po štirih urinih ciklih se prenese štiribitna informacija na izhode A_A do A_D. Serijsko vnešena informacija je sedaj na razpolago v paralelni obliki.

Če nam je dostopen le izhod A_D, lahko serijsko vnešeno informacijo v naslednjih treh urinih ciklih tudi serijsko preberemo na tem izhodu. Z nizkim logičnim signalom na krmilnem vhodu MC so vhodi za paralelni vnos E_A do E_D in urin signal T_2 zapahnjeni.

Če bi želeli vnašati podatke v obrnjeni smeri, tj. od E_D proti E_A, bi morali prevezati izhode FF-ov z vhodi v vezje in sicer: A_D z E_C, A_C z E_B in A_B z E_A. Serijski vnos gre preko E_D.

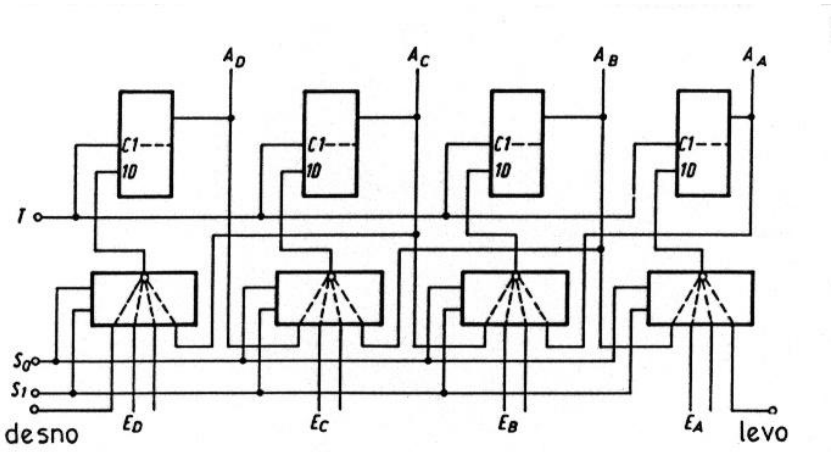
Po štirih urinih ciklih je informacija na izhodih. Pri tem mora biti MC v logičnem stanju 1, urin signal pa na vhodu T_2.

Vhodna informacija mora biti na vhodih FF-ov. Ker je vsakokrat neuporabljen urin signal s krmilnim vhodom MC zapahnjen, lahko oba vhoda T_1 in T_2 povežemo, če želimo obe smeri pomika (T_1 za levi premik, T_2 za desni pomik).

Paralelni vnos podatkov se vrši preko vhodov E_A do E_D ob taktu T_2. Krmilni vhod MC mora biti v visokem logičnem stanju. S tem so zapahnjeni vhod E_S, T_1 in vrata med FF-i. Pomik informacije v tem stanju krmilnih signalov ni možen.

Pri tem načinu delovanja se informacija iz vhodov prenese na izhod v enem urinem ciklu. Podatek ostane na razpolago, dokler se ne vpiše nova informacija (ali dokler se vsebina ne zbriše). Funkcija tega registra je na ta način razširjena še na »vmesno pomnenje«.

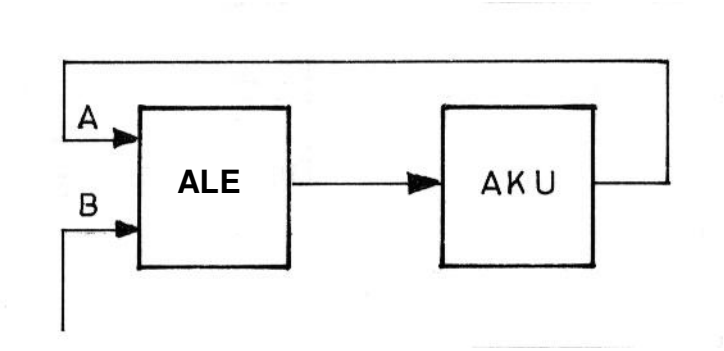
Na sl. 1.27 imamo podoben pomični register izveden z D-FF. Na vhodu vsakega D-FF je podatkovni multipleksor, ki ga krmilimo preko vhodov S_0 in S_1. Ta dva vhoda lahko izbirata med štirimi različnimi vhodi, ki se lahko prenesejo na izhode D-FF.



Slika: 1.27: Pomični register z D-FF

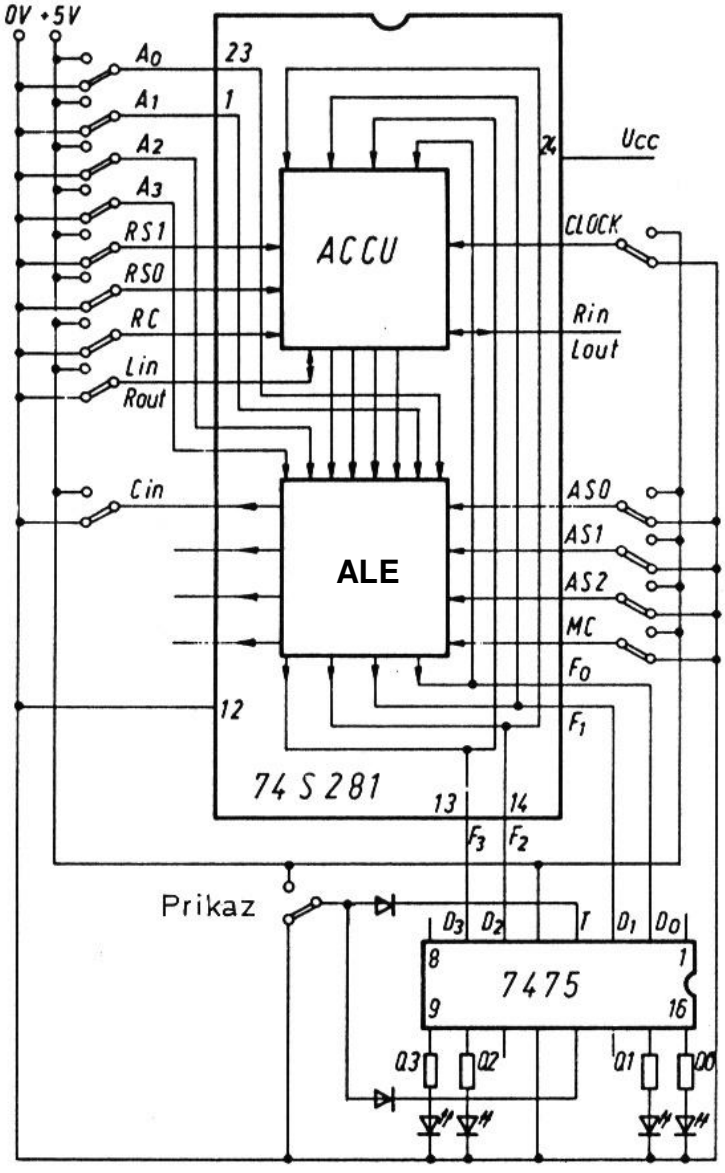
Tako lahko izbiramo med paralelnim vnosom podatkov iz E_D do E_A na vhode D-FF ali pa preko desnega vhoda serijsko z levim pomikom ali pa preko levega vhoda serijsko z desnim pomikom. En vhod je prost še za eno podatkovno besedo za paralelni vnos podatkov.

Tak pomični register, v funkciji akumulatorja, dopolnjuje aritmetično logično enoto in je prikazan na sliki 1.28. Akumulator služi za vmesno shranjevanje rezultatov. Pri tem vidimo, da so vhodi A povezani z izhodi iz registra. Tako je vodenje rezultatov nazaj na vhod ALE izvedeno preko akumulatorja. Podatki na vhodu B so lahko po aritmetični ali logični operaciji povezani z vsebino akumulatorja, rezultat se zopet shrani v akumulator, prejšnja vsebina pa se izgubi.



Slika 1.28: Pomični register v vlogi akumulatorja

Na sliki 1.29 je prikazana štiribitna računska enota, ki je sestavljena iz 4-bitnega akumulatorja, ALE enote (74S281) in je opremljena še z vhodno tastaturo, prikazovalno enoto in vmesnim pomnilnikom 7475.



Slika 1.29: Štiribitna računska enota

Primerjava slik 1.6 in 1.29 kaže že precejšen napredek v zmožnostih slednje. V nasprotju z ALE s slike 1.6 ima enota na sliki 1.29 le še en podatkovni vhod A_0 do A_3. Krmilne vhode S_0 do S_4 s slike 1.6 nadomeščajo na sliki 1.29 krmilni vhodi (Mode Control/MC/ AS_0, AS_1, AS_2 in C_{in} (vhod za inkrementiranje). Dodatno so na sliki 1.29 še krmilni vhodi akumulatorjev in sicer: urin signal (CLOCK), RS_0 (pomik v desno), RS_1 (pomik v levo), L_{in} (levi vhod), R_{out} (desni izhod), RC (Register Control namesto Mode Control), L_{out} (levi izhod), R_{in} (desni vhod). Vidimo tudi, da je izhod akumulatorja povezan z vhodom B ALE.

V naslednji tabeli so podane najbolj pomembne krmilne funkcije (od 32 možnih), koder vidimo, da se vse nanašajo na ALE in na akumulator.

Tabela 1.6: Krmilne funkcije ALE in akumulatorja na sliki 1.29

Krmilne funkcije						
AS0	**AS1**	**AS2**	**M**	**C_{in}**	**Funkcije**	**Okrajšava**
0	0	0	0	1	Vstavi AKU = -1	SM 1
1	0	0	0	1	Odštej A od AKU	SUB
1	1	0	0	0	Prištej A v AKU	ADD
0	0	1	0	1	Inkrementiraj AKU	INC
0	1	1	0	0	Polni A v AKU Polni B v AKU	LDA
0	0	0	1	X	Brez operacije	NOP
1	0	0	1	X	AKU EXKLUZIVNI-ALI A v AKU (A $\forall$ B)	XOR
0	0	1	1	X	AKU IN A v AKU (A $\wedge$ B)	AND
1	1	1	1	X	AKU ALI A v AKU (A $\vee$ B)	IOR

Iz tabele krmilnih funkcij lahko vidimo, kakšne operacije lahko izvaja vezje s sl. 1.29. Pri kombinaciji krmilnih funkcij AS_0 do C_{in} npr. 00101 izvaja vezje funkcijo B + 1. Ker je B zaradi povratne povezave vsakokrat vsebina akumulatorja, se s tem ukazom ob vsakem urinem ciklu vsebina akumulatorja poveča za 1. Pri takšni kombinaciji krmilnih signalov deluje akumulator kot števec. Okrajšave v desnem stolpcu, ki so splošno uporabljene v izmenjavi podatkov, so izpeljane iz angleškega opisa določene funkcije.

Zahtevnejše funkcije, ki bi jih s tem vezjem želeli izvajati, moramo najprej razbiti na enostavnejše. V ta namen je potrebnih več delovnih ciklov (urinih ciklov). Kot primer si oglejmo množenje števil 3 x 4. Ker vezje nima zmožnosti množenja, se moramo poslužiti večkratnega seštevanja.

Ta primer lahko rešimo z dvema krmilnima kombinacijama v treh programskih korakih.

1. korak	LDA	01100	Napolni akumulator z vsebino A
2. korak	ADD	11000	Prištej A k vsebini akumulatorja
3. korak	ADD	11000	Prištej A k vsebini akumulatorja

Kombinacija krmilnih funkcij prvega koraka povzroči, da se število 4, ki je na podatkovnih vhodih, ob prvem urinem ciklu prenese v akumulator. Potem se nastavi krmilna kombinacija drugega koraka (1100), ki v drugem urinem ciklu povzroči, da se podatek A, ki je še vedno na podatkovnem vhodu, prišteje k vsebini akumulatorja (A + B). Pri nespremenjeni kombinaciji

krmilnih signalov se operacija prištevanja A k vsebini akumulatorja v tretjem koraku pri naslednjem urinem ciklu še enkrat ponovi. Vsebina akumulatorja je sedaj
4 + 4 + 4 = 12, ki jo lahko s tipko *prikaz* prikažemo na izhodu.

V tabeli 1.7 so zbrane značilne funkcije akumulatorja glede na nivoje krmilnih vhodov.

Tabela 1.7: Značilne funkcije akumulatorja

Izbira funkcija (MC)			
Krmiljenje registra			
RSO	**RS1**	**RC**	**Opis funkcije**
0	0	X	akumulator je iz ALE polnjen paralelno
1	0	0	desni pomik (logičen)
1	0	1	desni pomik (aritmetičen); upošteva predznačni bit
0	1	0	levi pomik; (aritmetično) množenje; upošteva predznak
1	1	1	vsebina akumulatorja
X	X	X	nespremenjena

Poglejmo si progam, ki kaže možne krmilne funkcije ALE in akumulatorja v medsebojni odvisnosti:
V programu želimo odšteti štiribitno binarno število A od 4-bitnega binarnega števila B. Iznos naj preko inkrementiranja povečamo zopet na prvotno vrednost števila B. To vrednost želimo prikazati, jo potem deliti z 2 in jo zopet prikazati.

Dobimo naslednji potek programa:

Tabela 1.8: Primer izvajanja programa

Progr. korak	**Prikaz**	**Ura**	**Podatkovni vhod**				**ALU-funkcije**					**AKU-funkcije**			**Vsebina akumulatorja**			
			A_3	**A_2**	**A_1**	**A_0**	**AS_0**	**AS_1**	**AS_2**	**M**	**C_{in}**	**RS_0**	**RS_1**	**RC**	**Q_D**	**Q_C**	**Q_B**	**Q_A**
1.	0	↕	1	0	0	0	0	1	1	0	0	0	0	X	0	0	0	0
															1	0	0	0
2.	0	↕	0	0	1	1	1	0	0	0	1	0	0	X	1	0	0	0
															0	1	0	1
3.	0	↕	X	X	X	X	0	0	1	0	1	0	0	X	0	1	0	1
															0	1	1	0
4.	0	↕	X	X	X	X	0	0	1	0	1	0	0	X	0	1	1	0
															0	1	1	1
5.	1	↕	X	X	X	X	0	0	1	0	1	0	0	X	0	1	1	1
															1	0	0	0
6.	1	↕	X	X	X	X	0	0	0	0	0	1	0	1	1	0	0	0
															0	1	0	0

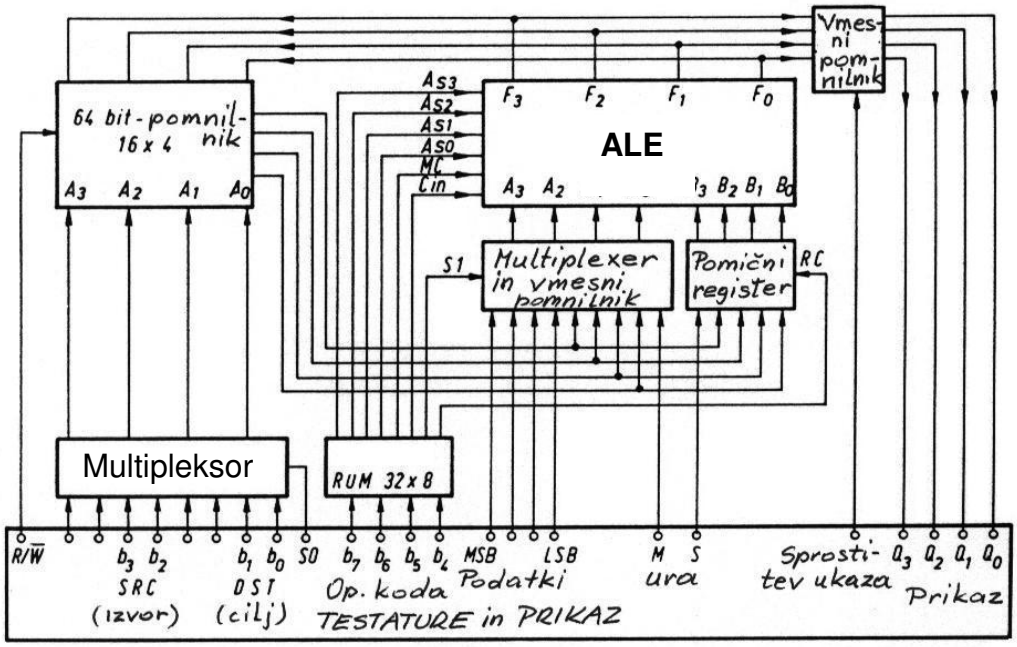
V prvem koraku pripravimo operand B = 1000 za ukaz LDA = 01100. Potem pritisnemo tipko CLOCK. S tem se ta operand vpiše v akumulator, ostane pa tudi na vhodu v ALE.

V drugem koraku pripravimo bitni vzorec operanda A = 0011 in bitni vzorec ukaza odštevanja SUB = 10001. Po pritisku tipke CLOCK dobimo v akumulatorju vmesni rezultat 0101.

Naslednji trije programski koraki povišujejo rezultat v akumulatorju vsakič za 1. Podatkovni vhod pri tem nima vpliva. Po treh inkrementih v petem koraku po pritisku tipke CLOCK je v akumulatorju, ki ga ves čas gledamo na prikazu, zopet vmesni rezultat 1000.

V šestem koraku se po pritisku tipke CLOCK vsebina akumulatorja pomakne za en bit v desno in prikaže se vrednost 0100. V šestih korakih smo izpeljali račun: (B-A + 1 +1 +1) : 2

Dopolnitev računalniškega vezja s slike 1.29 s spominom namesto akumulatorja, znatno razširi njegovo uporabnost. Na sliki 1.30 imamo računalniško vezje dopolnjeno z bralno-pisalnim spominom spominske kapacitete 64 bitov. Ta je razdeljen na 16 akumulatorjev po 4 bite. Sedaj lahko že govorimo o registrsko-aritmetični enoti. Gradnik 74281, ki vsebuje ALU in AKU na sliki 1.29 smo sedaj zamenjali z gradnikom, ki vsebuje le ALU in sicer 74181. Podatki se prenašajo v ALE preko multipleksorja z vmesnim pomnjenjem (74298). Pomični register 7495 lahko vrši pomične funkcije za vseh 16 podatkovnih besed v spominu. Preko vmesnega pomnilnika 7475 lahko prenašamo podatke za prikaz.



Slika 1.30: Vključitev bralno-pisalnega spomina v mikroračunalnik

V 16 akumulatorjih lahko hranimo 16 vmesnih rezultatov ali pa v njih shranimo kakšne druge informacije oziroma konstante.

V tabeli 1.9 je zbranih 16 najpomembnejših aritmetičnih in logičnih funkcij za ALE, multiplekser in pomični register 7495.

V desnem stolpcu pod »operacijo« so nekateri novi izrazi in sicer:
SRC - pomeni vsebino izvornega registra (Source Register)
DST - pomeni vsebino ciljnega registra (Destination Register)

Ta splošna definicija je potrebna zato, ker se lahko na vsaki od 16 spominskih besed izvaja ali aritmetična ali logična operacija.

Tabela 1.9 Aritmetične in logične operacije za vezje na sliki 1.30

Krmilne funkcije								**Operacija okrajšava**	**OPERACIJA**
AS_3	**AS_2**	**AS_1**	**AS_0**	**MC**	**C_{in}**	**MS**	**S_1**		
0	0	0	0	0	0	0	0	NOP	ni operacije
0	0	0	0	0	0	1	0	MOVE	vsebina SRC v DST
1	1	1	1	1	X	0	0	OUT	vsebina SRC na izhod
0	0	1	1	0	1	0	0	CLR	brisanje spominskega registra
0	0	1	1	0	X	0	0	(SET) SM 1	-1 v SRC
0	0	0	0	0	1	0	0	INC	vsebina SRC + 1
1	1	1	1	0	0	0	0	DEC	vsebina SRC -1
0	0	0	0	1	X	0	0	(COM) CMA	komplement SRC
1	0	0	1	0	0	1	0	ADDR	prištej: (DST)+(SRC)→DST
0	1	1	0	0	1	1	0	SUBR	odštej: (DST)-(SRC)→DST
1	1	1	0	1	X	1	0	IORR	poveži: (SRC)∨(DST)→DST
0	1	1	0	1	X	1	0	(EORR)XORR	poveži: (SRC)∨(DST)→DST
1	1	1	1	0	1	1	1	IN	podatkovni vhod v DST
1	0	1	1	1	X	1	0	ANDR	poveži(SRC)∧(DST)→DST
1	1	0	0	0	0	1	0	ASLR	vsebina registra eno mesto LEVO
1	0	1	0	1	X	0	0	ASRR	vsebina registra eno mesto DESNO

Funkcija MOVE v drugi vrstici pomeni na primer prenos vsebine določenega »izvornega« registra v nek drug »ciljni« register, torej od SRC v DST.

Poleg registrov in ALU imamo na naši sliki še dva dodatna gradnika. Dekodirnik v obliki 32 x 8 PROM-a, zreducira vseh 16 krmilnih funkcij z zgornje tabele na 4-bitno vhodno kodo. Selektor podatkov (74157) ali multipleksor, priredi naslove izvornih in ciljnih registrov po naslednjem vrstnem redu:

Tabela 1.10

Izvorni register SRC			**Ciljni register DST**		
b3	**b2**	**Št. registra**	**b1**	**b0**	**Št. registra**
0	0	Spom. register R0	0	0	Spom. register R0
0	1	Spom. register R1	0	1	Spom. register R1
1	0	Spom. register R2	1	0	Spom. register R2
1	1	Spom. register R3	1	1	Spom. register R3

Po definiciji registrov lahko poenostavimo še operacijske kode iz tabele 1.9:

Tabela 1.11

Operacijska koda				Ukaz	Funkcija
b7	**b6**	**b5**	**b4**		
0	0	0	0	NOP	(SRC)→SRC
0	0	0	1	MOVE	(SRC)→DST
0	0	1	0	OUT	(SRC)→IZHOD
0	0	1	1	CLR	0→SRC
0	1	0	0	SM1	-1→SRC
0	1	0	1	INC	(SRC)+1→(DST)
0	1	1	0	DEC	($\underline{\text{SRC}}$)-1→DST
0	1	1	1	CMA	(SRC)→DST
1	0	0	0	ADDR	(SRC)+(DST)→(DST)
1	0	0	1	SUBR	(DST)-(SRC)→(DST)
1	0	1	0	IORP	(SRC) $\vee$ (DST)→(DST)
1	0	1	1	XORR	(SRC) $\vee$ (DST) → (DST)
1	1	0	0	INR	(VHOD)→DST
1	1	0	1	ANDR	(SRC) $\wedge$ (DST)→DST
1	1	1	0	ASLR	(SRC) X 2^1→SRC
1	1	1	1	ASRR	(SRC) X 2^{-1} →SRC

Za prireditev nagovorjenega registra in za operacijsko kodo potrebujemo skupaj 8 bitov in sicer od b0 do b7.

V nekem računalniškem sistemu bi lahko to izgledalo tako, kot je prikazano spodaj:

Tabela 1.12

Operacijska koda				SRC		DST	
b7	**b6**	**b5**	**b4**	**b3**	**b2**	**b1**	**b0**

Operacijska koda NOP pomeni, da vsebina nekega registra ostane v tem registru. To pomeni, da se ne izvaja nobena operacija.

Kompleten ukaz v strojnem jeziku bi bil na primer:

0000	10	10	Skrajšan zapis
Op. koda	SRC	DST	NOP R2.R2

Ukaz prenese podatke iz R2 v R2. Vsebina iz R2 se je preko ALE prenesla v R2, torej se ni nič zgodilo.

Operacijska koda 0001 je ukaz MOVE (prenesti), ki prenaša podatke med akumulatorskimi registri. Če želimo na primer prenesti vsebino registra RO v register R3, se glasi ukaz:

0001	00	11	Skrajšana oblika
Op. koda	SRC	DST	MOVE R0, R3

Vsebina izvornega registra se pri tem ne izgubi. Nekateri mikroprocesorski sistemi pri takšnem ukazu pomikajo podatke od desne proti levi. Ukaz MOVE bi se tedaj glasil takole:

MOVE R3, R0

Vsi ukazi nabora na naši listi imajo podobne funkcije: npr. Ukaz ADDR (R-pomeni registrski ukaz) z operacijsko kodo 1000 prišteva vsebino SRC registra v DST register. Puščica pomeni, da je suma v DST registru.

Ukaz SUBR (odštevanje) z operacijsko kodo 1001 odšteva vsebino SRC od vsebine DST. Pri tem ukazu se vsebina SRC odšteje od DST in ne obratno, čeprav je v tabeli to videti rahlo nesimetrično, nudi ta vrstni red nekatere praktične prednosti in je zato uporabljen v mnogih mikroprocesorjih.

Za zaključek obravnave »registrsko aritmetične« enote izpeljimo kratek program. Vsebini RO in R1 bomo sešteli in od iznosa odšteli vsebino R2.

Vmesni rezultat bomo nato delili z 2 in potem prenesli v register R3.

Program izgleda takole:

Tabela 1.13

Programski korak	**Strojna koda**			**Ukaz Skrajšano**
1.	1100	XX	00	INR (VHOD V R0)
2.	1100	XX	01	INR (VHOD V R1)
3.	1100	XX	10	INR (VHOD V R2)
4.	1000	00	01	ADD, R0, R1
5.	1001	10	01	SUBR R1, R2
6.	1111	01	XX	ASRR R1
7.	0001	10	11	MOVE R1, R3

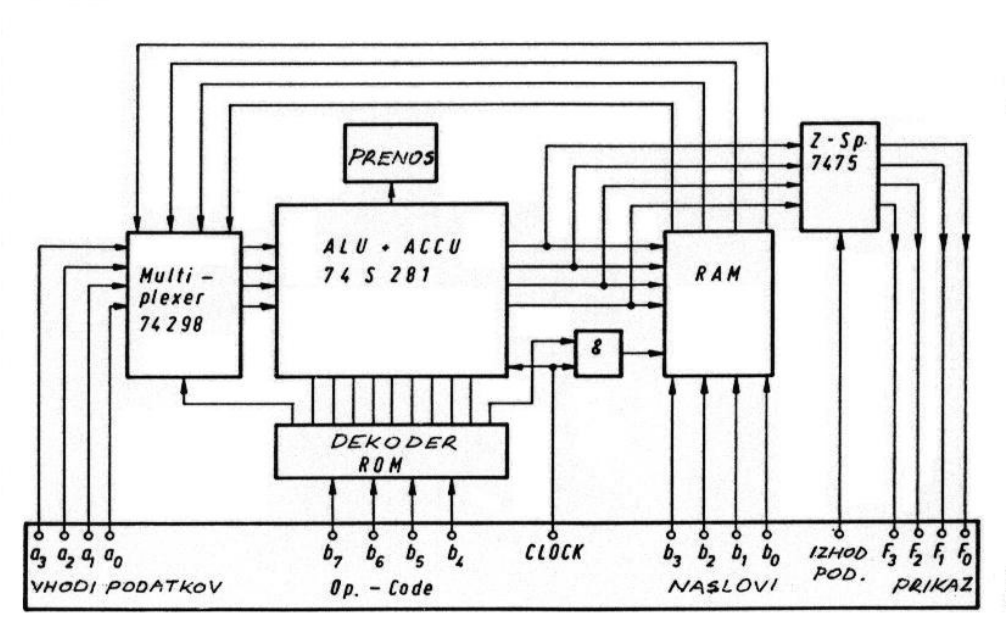
Izvajanje programa poteka takole:

1. Nastavitev prve strojne kode in števila 0010 na podatkovnem vhodu
   Pritisniti CLOCK M
2. Nastavitev druge strojne kode in števila 1000 na podatkovnem vhodu
   Pritisniti CLOCK M
3. Nastavitev tretje strojne kode in števila 0100 na podatkovnem vhodu
   Pritisniti CLOCK M
4. Nastavitev četrte strojne kode
   Pritisniti CLOCK M
5. Nastavitev pete strojne kode
   Pritisniti CLOCK
6. Nastavitev šeste strojne kode
   Pritisniti CLOCK
7. Nastavitev sedme strojne kode
   Pritisniti CLOCK.

Spominske registre v glavnem uporabljamo za shranjevanje vmesnih ukazov in manj za podatke. Za shranjevanje podatkov se uporablja bralno pisalni spomin (RAM).

Dokompletirajmo sedaj naše prejšnje vezje z nekaterimi spremembami še s podatkovnim spominom.

Zaradi poenostavitve se poslužimo kombinacije obeh. Takšno modificirano vezje je prikazano na sliki 1.31.



Slika 1.31: Vključitev RAM spomina k ALE

Za izvajanje funkcij ALE in AKU je privzet gradnik 74S281 s slike 1.28. Kod dekodirnik krmilnih funkcij je uporabljen ROM s slike 1.30. Kot selektor podatkov je izbran multipleksor 74298 s slike 3.30. Kot bralno pisalni spomin je uporabljen RAM pomilnik 32 x 32 za vmesno pomnjenje na izhodu je zopet gradnik 7475 iz slike 3.30. Dodan je še »Flag« za registriranje aritmetičnega prenosa in skupni taktni vhod (CLOCK).

To vezje, ki pa je vodljivo ročno in še ne avtomatsko, že ima osnovne lastnosti mikroprocesorskih funkcij s podatkovnim pomnilnikom.

V nasprotju z vezjem na sliki 1.30, kjer je bilo možno vmesne rezultate nalagati v 16 spominskih registrov, pa vezje na sliki 1.31 omogoča shranjevanje v bralno-pisalni spomin poleg vmesnih rezultatov tudi konstante in operande, ki jih lahko uporabi pri kasnejših operacijah.

Multipleksor na vhodu aritmetično-registrskega vezja lahko prepusti podatke z vhoda ali pa iz spomina. Vsebina akumulatorja se lahko prenese v spomin ali vsebina iz spomina preko ALE v akumulator. Multipleksor je voden preko ROM dekodirnika.

Če na vhodih b4 do b7 ni nobene operacijske kode, je bralno-pisalni spomin povezan z aritmetično registrsko enoto preko multipleksorja. Operacijska koda 1101 (INP) preklopi multipleksor na podatkovne vhode, tako da se lahko vhodni podatki prenesejo preko ALE v akumulator. V ALE se podatki pri tem procesu ne spremenijo.

Neka druga operacijska koda s spominskim naslovom bo sprostila podatke iz neke spominske lokacije. Operacijska koda 1110 sprosti spomin za vpis podatkov iz akumulatorja. Vsebina akumulatorja se pri tem ne izgubi.

Spodnja tabela kaže najpomembnejše funkcije vezja s slike 1.31. So identične tistim v funkcijski tabeli, ki velja za sliko 1.30. Pri nekaterih ukazih podatkovni spomin ni uporabljen, zato spominske naslovi b_0 do b_3 nimajo pomena in lahko zavzamejo poljubne vrednosti.

Tabela 1.14

Operacijska koda b_7 b_6 b_5 b_4	**Spominske Adrese** b_3 b_2 b_1 b_0	**Ukaz**	**Flag vstavljen**	**FUNKCIJA**
0 0 0 0	X X X X	NOP	1	ni operacije
0 0 0 1	X X X X	SP1	1	vstavi AKKU = 1
0 0 1 0	X X X X	CMA	0	komplementiraj AKKU
0 0 1 1	b b b b	LDA	0	napolni vsebino (naslov bbbb)
0 1 0 0	X X X X	CLA	0	briši AKKU
0 1 0 1	X X X X	INC	1	vsebina AKKU+1
0 1 1 0	X X X X	DEC	1	vsebina AKKU-1
0 1 1 1	b b b b	ADD	1	prištej vsebino (bbbb)
1 0 0 0	b b b b	SUB	1	odštej vsebino (bbbb)
1 0 0 1	b b b b	AND	1	AKKU $\wedge$ vsebina (bbbb)
1 0 1 0	b b b b	IOR	1	AKKU $\vee$ vsebina (bbbb)
1 0 1 1	b b b b	XOR	1	AKKU $\forall$ vsebina (bbb)
1 1 0 0	X X X X	SM1	1	vstavi AKKU-1
1 1 0 1	X X X X	INP	0	polni podatke v AKKU
1 1 1 0	b b b b	STA	0	shrani AKKU v (bbbb)

Podatki se vpisujejo v spomin z ukazi 1101 (INP) in 1110 (STA). Ker ima spomin štiri naslovne linije, lahko z njim naslavljamo 16 spominskih lokacij in jih napolnimo s podatki.

V naslednjem primeru napolnimo 16 spominskih lokacij s podatki 0000 do 1111.

Tabela 1.15

Naslov				**Podatki**			
b_3	**b_2**	**b_1**	**b_0**	**a_0**	**a_1**	**a_2**	**a_3**
0	0	0	0	1	1	1	1
0	0	0	1	1	1	1	0
0	0	1	0	1	1	0	1
0	0	1	1	1	1	0	0
0	1	0	0	1	0	1	1
0	1	0	1	1	0	1	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	0	0
1	0	0	0	0	1	1	1
1	0	0	1	0	1	1	0
1	0	1	0	0	1	0	1
1	0	1	1	0	1	0	0
1	1	0	0	0	0	1	1
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	1
1	1	1	1	0	0	0	0

Pri tem je potreben naslednji potek programa:

1. Korak:
   Podatkovne vhode a_3 do a_0 nastavimo na 1111. Operacijsko kodo b_7 do b_4 nastavimo na 1101 (INP). Pritisnemo tipko CLOCK. Podatki so v akumulatorju.
2. Korak:
   Operacijsko kodo in naslov nastavimo na 11100000. Pritisnemo tipko CLOCK. Podatki se prenesejo iz akumulatorja v spomin na naslov 0000.
3. korak:
   Podatkovne vhode a_3 do a_0 nastavimo na 1110. Operacijsko kodo b_7 do b_4 nastavimo na 1101. Pritisnemo tipko CLOCK. Podatki so zopet v akumulatorju.
4. Korak:
   Operacijsko kodo in naslov nastavimo na 11100001. Pritisnemo tipko CLOCK.

Nadaljujemo v smislu 1 in 2 ali 3 in 4.

Podatki ostanejo v spominu dokler ne pride do novega ukaza STA ali dokler ne izpade napajanje.

V naslednjem primeru vzemimo besedo iz neke spominske lokacije in jo vnesimo v akumulator.

1. korak: Operacijsko kodo in naslov nastavimo na 00110110. Pritisnemo tipko CLOCK.

Vsebina lokacije 0110, ki je 1001, je sedaj v akumulatorju in v spominski lokaciji, ker se pri branju ni izgubila. Pritisnemo tipko za prikaz.

Vsebina akumulatorja (1001) se prikaže. V naslednjem primeru bomo vsebino lokacije 1010 (0101) odšteli od vsebine na lokaciji 0000 (1111).
V prvem programskem koraku nastavimo ukaz LDA (operacijska koda 0011 in naslov 0000) in pritisnemo tipko CLOCK, da se prenese vsebina na naslovu 0000 v akumulator.

V drugem koraku nastavimo ukaz SUB (op. Koda 1000, naslov 1010) in s tipko CLOCK se izvrši operacija odštevanja. Vsebina lokacije 1010 se odšteje od vsebine akumulatorja 1111. Rezultat 1010 kot nova vsebina akumulatorja se lahko prikaže, če pritisnemo tipko prikaz.

V zadnjem primeru naredimo logično AND operacijo med podatkoma spominske lokacije 0011 in podatkovnim vhodom.

V prvem koraku nastavimo podatke (0101) in ukaz INP (op. Koda 1101). S pritiskom CLOCK se podatki prenesejo v akumulator.

V drugem koraku nastavimo ukaz AND in naslov spominske lokacije (op. Koda 1001, naslov 0011). S pritiskom na CLOCK povežemo po logični funkciji AND podatke na dotičnem naslovu v spominu (naslov 0011, podatki 1100) in vsebino akumulatorja.

Vsebina akumulatorja		0101
Vsebina spominske lokacije		1100
	AND	0100 – nova vsebina akumulatorja

To lahko vidimo s pritiskom tipke »Prikaz«.

Pri teh primerih smo lahko videli, da se aritmetične in logične operacije ADD, SUB, AND, IOR in XOR izvajajo med vsebino naslavljane spominske lokacije in vsebino akumulatorja.

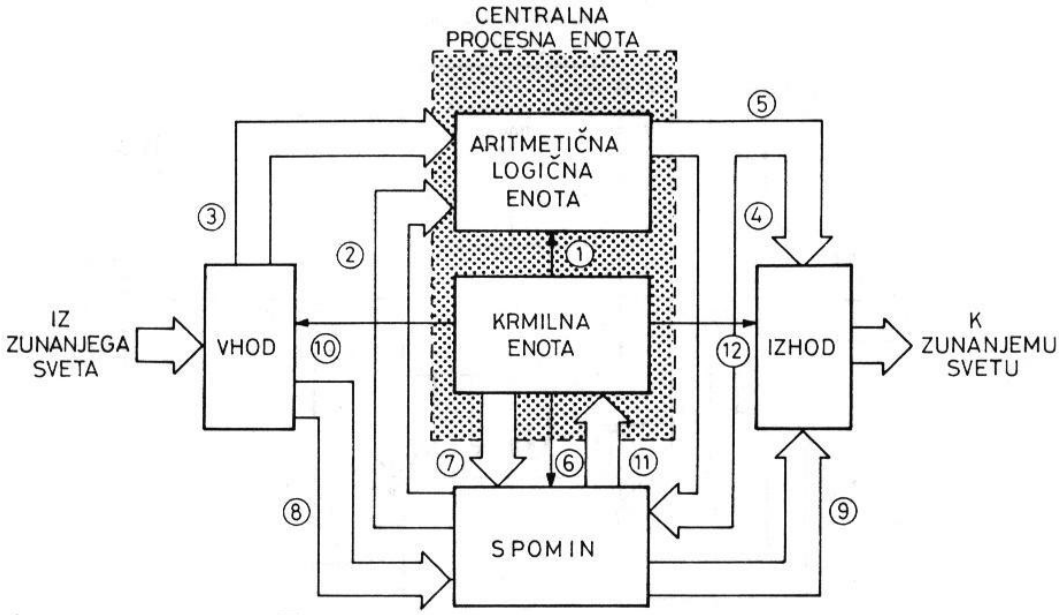
Vsebina akumulatorja se lahko oblikuje z ukazom LDA iz spomina ali pa z ukazom INP s podatkovnega vhoda. Pri LDA ukazu so brez pomena podatkovni vhodi pri INP ukazu pa spominski naslovi.

1.5 Osnovna organizacija klasičnega mikroračunalnika

Vsak mikroračunalnik vsebuje pet osnovnih sestavnih elementov ali enot:

1. Aritmetična logična enota (ALE, ang. ALU).
2. Spominska enota.
3. Krmilna enota (ang. Control Unit).
4. Vhodna enota (ang. Input Unit).
5. Izhodna enota (ang. Output Unit).

Osnovna medsebojna povezava teh enot je prikazana na sliki 1.32. Puščice v tem diagramu kažejo smeri, v katerih tečejo podatki, informacije ali krmilni signali. Imamo dve različni širini puščic: široke puščice ponazarjajo povezave za podatke in informacije, ki zasedajo običajno večje število paralelnih linij in tanjše puščice, ki ponazarjajo krmilne signale. Širše puščice so tudi numerirane zaradi nadaljne lažje razlage in opisa.



Slika 1.32: Najpomembnjši deli mikroračunalnika

1.5.1 Aritmetična logična enota – ALE

ALE je del mikroračunalnika v katerem se izvajajo aritmetične in logične operacije s podatki. Tip operacije, ki se izvaja, je določen s krmilno enoto (puščica 1). Podatek, na katerem naj se operacija ALE izvaja, lahko pride iz spominske enote (puščica 2) ali pa iz vhodne enote (puščica 3). Rezultat operacije, ki je bil dosežen v ALE, je lahko prenešen v spominsko enoto (puščica 4) ali pa v izhodno enoto (puščica 5).

1.5.2 Spominska enota

Spominska enota shranjuje skupine binarnih besed, ki so lahko instrukcije, po katerih naj mikroračunalnik deluje ali pa podatki, ki se uporabljajo med izvajanjem programa. Spominska enota služi tudi za shranjevanje vmesnih in končnih rezultatov aritmetičnih operacij (puščica 4). Krmilna enota določa ali gre za bralno ali pa za pisalno operacijo v spominu (puščica 6). Lokacijo v spominu določa krmilna enota z naslovom (puščica 7). Podatek je lahko vpisan v spomin iz ALE ali pa iz vhodne enote (puščica 8) pod nadzorom krmilne enote. Podatek se lahko prenese iz spomina v ALE (puščica 2) ali pa v izhodno enoto (puščica 9).

1.5.3 Vhodna enota

Vhodna enota lahko obsega vse naprave, ki so namenjene zajemanju podakov, ki se posredujejo spominski enoti (puščica 8) ali pa ALE (puščica 3). Krmilna enota določa kam se bo posredoval podatek. Vhodna enota služi za vnos programa in podatkov v spominsko enoto preden mikroračunalnik starta. Ta enota prav tako služi za vnos podatkov v ALE iz zunanjih enot med izvajanjem programa. Nekatere vhodne enote so npr.: tastatura, A/D pretvorniki, periferni vmesniki, itd.

1.5.4 Izhodna enota

Izhodne enote so naprave, ki služijo za prenos podatkov iz mikroračunalnika v zunanji svet. Te izhodne enote se aktivirajo preko krmilne enote (puščica 12) in lahko sprejemajo podatke iz spomina (puščica 9) ali iz ALE (puščica 5), ter jih nato spremenijo v primerno obliko za zunanjo uporabo. Primeri izhodnih enot so LED ali LCD prikazovalnik, svetlobni indikator, monitor, D/A pretvorniki itd.

1.5.5 Krmilna enota

Iz dosedaj opisanega je delovanje krmilne enote sedaj že nekoliko bolj jasno. Usmerja delovanje vseh ostalih enot preko krmilnih signalov, ki so časovno pogojeni. V prispodobi je krmilna enota kot dirigent v orkestru, ki je odgovoren za soglasje vseh instrumentov in za njihovo sinhronizacijo. Ta enota vsebuje logična in časovna vezja, ki generirajo potrebne signale za izvrševanje vseh instrukcij v programu.
Krmilna enota dobi ukaz (instrukcijo) iz spomina na ta način, da vanj pošlje naslov (puščica 7) in bralni ukaz oz. impulz (puščica 6). Instrukcijska beseda, ki se nahaja na določenem naslovu v spominu, se potem prenese v krmilno enoto (puščica 11). Ta instrukcijska beseda, ki je v obliki neke binarne kode, se potem dekodira v logični vezjih krmilne enote. Na podlagi te informacije krmilna enota generira potrebne krmilne signale za njeno izvršitev.

1.5.6 Centralna procesna enota – CPE (ang. Central Process Unit – CPU)

Na sliki 1.32 sta ALE in krmilna enota podani skupaj v eni enoti, ki se imenuje centralna procesna enota (CPE). To je podano skupaj zato, da se loči dejansko jedro mikroračunalnika od drugih enot. Tak sklop dobimo integriran v enem ti. mikroprocesorskem čipu.

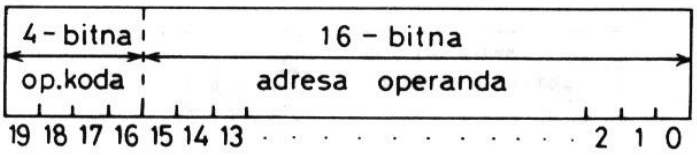
1.6 Mikroračunalniške besede

Dosedanja razlaga o delovanju posameznih enot v mikroračunalniku je bila precej poenostavljena. Za razumevanje mnogih podrobnosti moramo ločiti posamezne oblike informacij, ki se prenašajo in s katerimi manipulira mikroračunalnik.
Najbolj elementarna enota informacije je binarni digit (bit). Za praktično uporabo pa so bolj zanimive večbitne besede. Z dolžino besede se delno že določi zmogljivost mikroračunalnika. Npr. pri 16 bitnem mikroračunalniku se podatki in instrukcije procesirajo v obsegu 16 bitov. Dolžina besede označuje dolžino besede spominske enota. 16-bitni mikroračunalnik ima spominsko enoto, ki lahko shrani določeno število 16-bitnih besed. Večji mikroračunalniki imajo dolžino besed med 16 in 64 bitov.

1.6.1 Instrukcijske besede

Oblika, ki se uporablja za podatkovne besede, se zelo malo razlikuje med posameznimi vrstami mikroračunalnikov, razen v sami dolžini besed. To pa ne velja za instrukcijske besede. Te besede vsebujejo informacije, ki jih mikroračunalnik potrebuje za izvajanje različnih operacij. Oblike in kode teh besed pa se lahko različne med posameznimi izvedbami mikroračunalnikov. Za večino mikroračunalnikov obsega instrukcijska beseda dve enoti

informacije: operacija, ki naj se izvaja in naslov operanda (podatka) na katerem naj se ta operacija izvaja.
Na sliki 1.33 imamo prikazan primer enonaslovne instrukcijske besede za nek namišljen 20-bitni mikroračunalnik. 20 bitov instrukcijske besede je razdeljenih na dva dela.



Slika 1.33: Enostavna instrukcijska beseda

Prvi del instrukcijske besede (biti 16 do 19) vsebujejo operacijsko kodo ali ukaz. Ta 4-bitna operacijska koda določa operacijo, ki naj bi jo mikroračunalnik izvedel. (npr. seštevanje in odštevanje). Drugi del instrukcije (biti 0 do 15) tvori naslov operanda v spominu. Če ima operacijska koda 4 bite, lahko zapišemo $2^4 = 16$ različnih operacijskih kod. To pomeni, da bi tak mikroračunalnik lahko izvajal največ 16 različnih operacij. Instrukcijska beseda s slike 1.33 ima 16 bitov rezerviranih za naslove operandov. To pomeni, da ima $2^{16} = 65536$ možnih naslovov. Ta instrukcijska beseda lahko določa 16 različnih ukazov in 65536 naslovov operandov.

Primer:

Vzemimo, da operacijska koda 0100 določa seštevanje (ADD). Naslovna koda je v heksadecimalnem zapisu 5A72. V heksadecimalnem zapisu ima celotna instrukcijska beseda obliko:

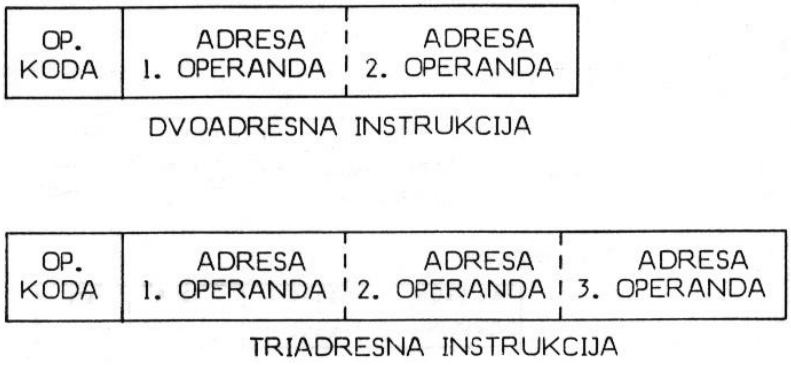
45A72

Takšna instrukcijska beseda narekuje mikroračunalniku naslednjo operacijo:

Preberi podatek, ki je shranjen na naslovu 5A72 in ga pošlji v ALE ter prištej številu, ki se nahaja v akumulatorju. Rezultat operacije naj se shrani v akumulatorju (prejšnja vsebina je izgubljena).

1.6.2 Instrukcije z več naslovi

Instrukcija z enim naslovom je najpreprostejša oblika, ki se uporablja v mikroračunalnikih. Uporabljamo tudi drugačne oblike instrukcij, ki vsebujejo več informacij v eni instrukcijski besedi.
Na sliki 1.34 imamo primer instrukcije z dvema naslovoma in instrukcije s tremi naslovi.



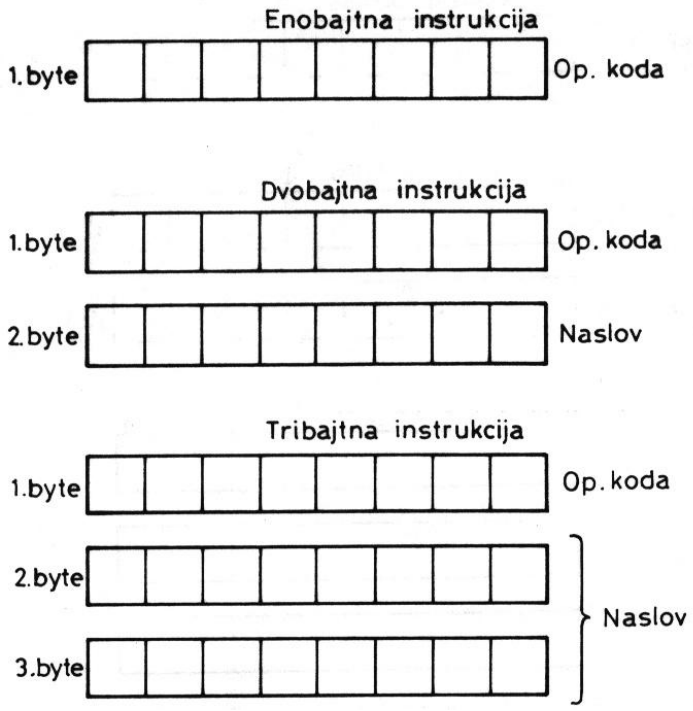
Slika 1.34: Dvo- in trinaslovna instrukcija

Instrukcija z dvema naslovoma vsebuje naslova dveh operandov, na katerih naj se operacija izvaja. Instrukcija s tremi naslovi ima poleg omenjenih dveh še tretji naslov, na katerega se shranjuje rezultat izvedene operacije. Seveda pa daljša instrukcijska beseda zahteva spominsko enoto z daljšo dolžino besede.

1.6.3 Instrukcije z več bajti

Prej smo obravnavali obliko instrukcijske besede, ki je vsebovala operacijsko kodo in naslov operanda v eni sami besedi. Takšna instrukcija je lahko shranjena v eni sami spominski lokaciji. To je tipično za mikroračunalnike z relativno dolgo spominsko besedo. Pri mnogih mikroračunalnikih pa to ni mogoče. Pri 8-bitnih mikroračunalnikih je dolžina besede 1 byte in imamo tri osnovne oblike formata, ki so prikazane na sliki 1.35:

- enobajtne,
- dvobajtne in
- tribajtne.



Slika 1.35: Instrukcijske besede pri osembitnem mikroračunalniku

Enobajtna instrukcija vsebuje samo 8 bitov operacijske kode brez naslova. Te instrukcije niso namenjene operacijam na operandih. Primer takšne instrukcije je npr. CLA (Clear Accumulator Register to Zero), ki ukazuje, naj se brišejo vsi FF registra (akumulatorja) v ALE.
Prvi bajt dvobajtne instrukcije je operacijska koda. Drugi bajt pa je 8-bitni naslov, ki določa lokacijo operanda. V tribajtni instrukciji pa drugi in tretji bajt formirata 16-bitni naslov operanda. Pri večbajtnih instrukcijah so bajti (dva ali trije), ki oblikujejo celotno instrukcijo, shranjeni v zaporednih spominskih lokacijah. To je spodaj ilustrirano za tribajtno instrukcijo.

Tabela 1.16: Primer izvajanja tribajtne instrukcije

Naslov v spominu (Heks.)	**Spominska beseda**		**Opis**
	Binarno	**Heks.**	
0020	01001010	4A	Operacijska koda
0021	00110101	35	Zgornja vrsta naslovnih bitov
0022	11110110	F6	Spodnja vrsta naslovnih bitov
.	.	.	
.	.	.	
35F6	01111100	7C	Operand

Levi stolpec podaja naslove lokacij v spominu, kjer so shranjeni vsi bajti. Ti naslovi so podani heksadecimalno. Drugi stolpec podaja binarne besede, ki so dejansko shranjene v spominu. Tretji stolpec je njihov heksadecimalni ekvivalent. Trije bajti, ki so shranjeni na lokacijah 0020, 0021 in 0022, sestavljajo kompletno instrukcijo za prištevanje podatkovne besede, ki je shranjena na naslovu 35F6 k vsebini akumulatorja. Drugi in tretji bajt vsebujeta 8 zgornjih in

8 spodnjih bitov naslova operanda. Nekateri mikroračunalniki uporabljajo obraten vrstni red tako, da spodnjo vrsto bitov podajajo v drugem bajtu, zgornjo pa v tretjem.
Spominska lokacija je tako določena; njena vsebina je podatkobna beseda, ki jo krmilna enota računalnika prenese v ALE za operacijo odštevanja.

1.6.4 Primer enostavnega programa

Ko smo pogledali tipe podatkovnih in instrukcijskih kod, je naslednji korak kombinacija podatkov in besed v programu. Za naše programiranje uporabimo fiktiven računalnik, ki lahko izvaja 16 instrukcij in ima 16 besed v spominu. To je seveda veliko manj kot pri običajnih mikroračunalnikih, vendar deluje na enak način kot večji. Oblika njegove instrukcijske besede je naslednja:

7	6	5	4	3	2	1	0
Op. koda				Naslov operanda			

Računalnik ima torej $2^4 = 16$ različnih možnih instrukcij. 4-bitni naslov operanda določa 16 lokacij v spominu. Spomin mora obsegati 16 8-bitnih besed.

V tabeli 1.17 je opisanih nekaj instrukcij tega mikroračunalnika. Vsaka je določena s 4-bitno operacijsko kodo ter s simbolično kodo, ki je sestavljena iz treh ali štirih črk (ang. Mnemonic). Simbolične kode si namreč lažje zapomnimo. Opisane instrukcije so zelo splošno uporabljene pri večini mikroračunalnikov. Operand je podatkovna beseda, ki je shranjena na naslovu v spominu.

Tabela 1.17: Nabor ukazov enostavnega mikroračunalnika

Simbolična koda	Binarna koda	Razlaga
LDA	1100	LOAD akumulator: podatki, ki so shranjeni na naslovu operanda, se prenesejo v akumulatorski register.
ADD	0100	ADD: operand se prišteje k številu v akumulatorju in rezultat se shrani v akumulator.
SUB	0101	SUBTRACT: operand se odšteje od vsebine akumulatorja in rezultat se shrani v akumulatorju.
STA	0111	STORE akumulator: vsebina akumulatorja se shrani v spominski lokaciji, ki je določena z naslovom operanda.
JPZ	1001	JUMP ON ZERO: naslednja instrukcija je vzeta iz naslova v spominu, če je vsebina akumulatorja 0. Sicer se izvede naslednja instrukcija po vrsti.
JMP	1000	JUMP (brezpogojno): naslednja instrukcija je vzeta z naslova v spominu ne glede na sekvenco.
HLT (STOP)	0001	Delovanje mikroračunalnika je ustavljeno.

Ob uporabi nekaj instrukcij iz zgornje tabele bomo napisali enostaven program, ki naj opravi sledečo nalogo:

1. Odštevanje števila (X) od (Y).
2. Shranjenje rezultata Q na lokaciji $1001 = 9_{16}$.
3. Če je rezultat 0, naj se izvajanje programa ustavi na lokaciji 0101 (5_{16}). Sicer naj se ustavi na lokaciji 0100 (4_{16}).

Program, ki je shranjen v spominu, je zapisan v tabeli 1.18. 16 spominskih lokacij je označenih z naslovi od 0 do F. Besede, shranjene na lokacijah od 0 do 5, so instrukcijske besede programa. Lokacije 7, 8 in 9 so uporabljene za podatkovne besede. Števili X in Y sta podatka, ki sta potrebna za izvajanje programa in sta shranjena na lokacijah 7 in 8. Lokacija 9 je rezervirana za hranjenje rezultata Q. Lokacija 6 in lokacije od A do F pri tem enostavnem programu niso uporabljene.

Tabela 1.18: Primer programa

Naslov v spominu	Spominska beseda (bin)	Simbolična koda	Opis
0	11001000	LDA 8	LOAD Y v akumulator
1	01010111	SUB 7	SUBTRACT X od akum.
2	01111001	STA 9	Shrani rezultat Q na lokaciji 9
3	10010101	JPZ 5	Če Q = 0, JUMP na 5
4	00010000	HLT	Če Q = 0, ustavi tukaj.
5	00010000	HLT	Če Q = 0, ustavi tukaj.
6	Ni uporabljeno		
7	$X_7, X_6, X_5, X_4, X_3, X_2, X_1, X_0$	X	
8	$Y_7, Y_6, Y_5, Y_4, Y_3, Y_2, Y_1, Y_0$	Y	
9	?	Q	Lokacija kjer naj se shrani Q
A - F	Niso uporabljene		

Izvajanje programa:

Ko je program shranjen v spominu, se lahko začne izvajati:

1. Delovanje se začne s tipko START ali RUN. To povzroči, da krmilna enota (CONTROL) začne z jemanjem instrukcij iz spomina. Prva instrukcija je LDA 8, ki omogoči prenos instrukcije s spominskega naslova 0 v krmilno enoto. To je LDA 8, ki naroča krmilni enoti, da pobere podatkovno besedo na spominski lokaciji 8 in jo prenese v akumulator. Podatkovna beseda na naslovu 8 je Y.
2. Po izvršeni operaciji na naslovu 0, krmilna enota avtomatsko preskoči na naslov 1 po svojo naslednjo instrukcijo. Za to zaporednost sprejemanja instrukcij skrbi programski števec (PC – Program Counter), ki starta pri 0 in se mu ob vsaki instrukciji stanje poveča za 1 tako, da vsakič vsebuje naslov naslednje instrukcije. Instrukcija na naslovu 1 je SUB 7 in naroča krmilni enoti, naj gre na naslov 7 po podatek X. Rezultat te operacije je v akumulatorju.
3. Vrednost PC se poveča za 1 in tvori naslov naslednje lokacije v spominu, kjer se nahaja instrukcija. Ta instrukcija (STA 9) narekuje krmilni enoti vpis vsebine akumulatorja na naslov 9 v spominu. Rezultat Q = Y – X je sedaj v spominski lokaciji 9, kakor tudi v akumulatorju.
4. Vrednost programskega števca se poveča na vrednost 3. Na naslovu 3 dobi krmilna enota naslednjo instrukcijo. JPZ 5 naroča krmilni enoti, da povpraša po vsebini akumulatorja. Če ni enako 0, krmilna enota poveča vrednost programskega števca na 4. Če je enako 0, pa se vrednost programskega števca poveča na 5. Tako je glede na vrednost Q, naslednja instrukcija prebrana na naslovu 4 ali 5.
5. Če je v programskem števcu vrednost 4, vzame krmilna enota instrukcijo iz lokacije 4 v spominu, ki povzroči zaustavitev izvajanja programa. Če pa je vrednost programskega števca 5, krmilna enota izvaja instrukcijo na naslovu 5, ki prav tako povzroči ustavitev delovanja mikroračunalnika.

6. Program se ne izvaja, dokler ga ne startamo ponovno ali pa ga napoti v izvajanja drugega programa.

1.7 Delovni cikli mikroračunalnika

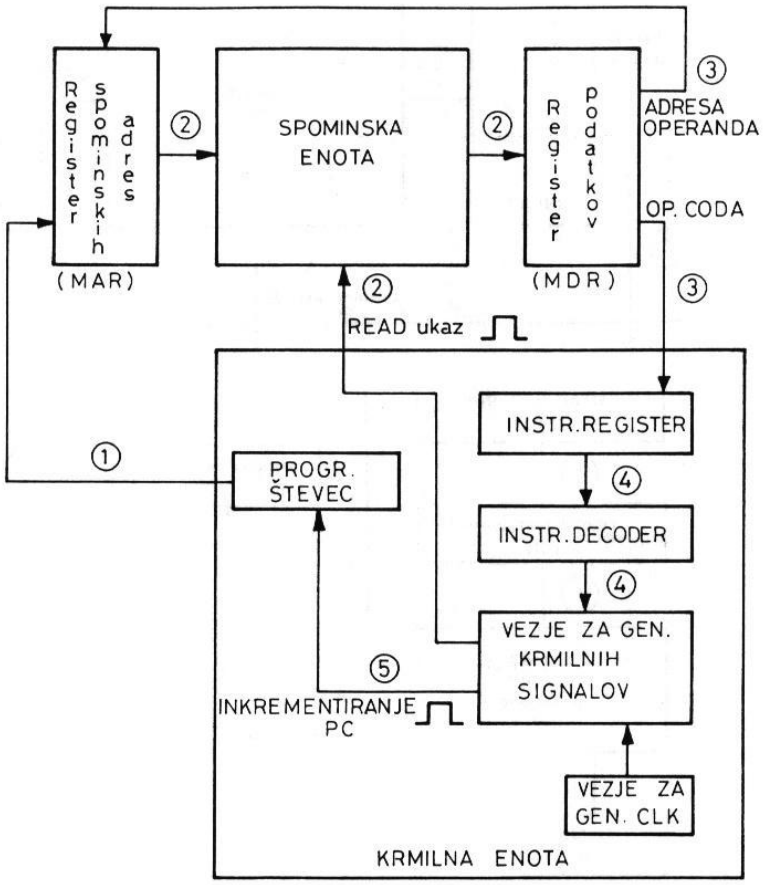
Sedaj nam je poznana osnovna programska sekvenca. V principu pa mikroračunalnik vedno izvaja eno od dveh operacij:

1. Sprejem instrukcijske besede iz spomina ter interpretacija instrukcije ali
2. Izvajanje operacije, ki jo narekuje instrukcijska beseda

Operacija mikroračunalnika je sestavljena torej iz dveh ciklov: instrukcijski cikel in cikel izvajanja. Poglejmo potek informacij med posameznimi enotami mikroračunalnika med potekom izvajanja teh dveh ciklov.

1.7.1 Instrukcijski cikel

Na sliki 1.36 je prikazana blokovna shema mikroračunalniških enot, ki so vključene v izvajanje instrukcijskega cikla. Puščice v shemi enostavno podajajo smer pretoka informacij. Označene številke podajajo korake, ki si sledijo.



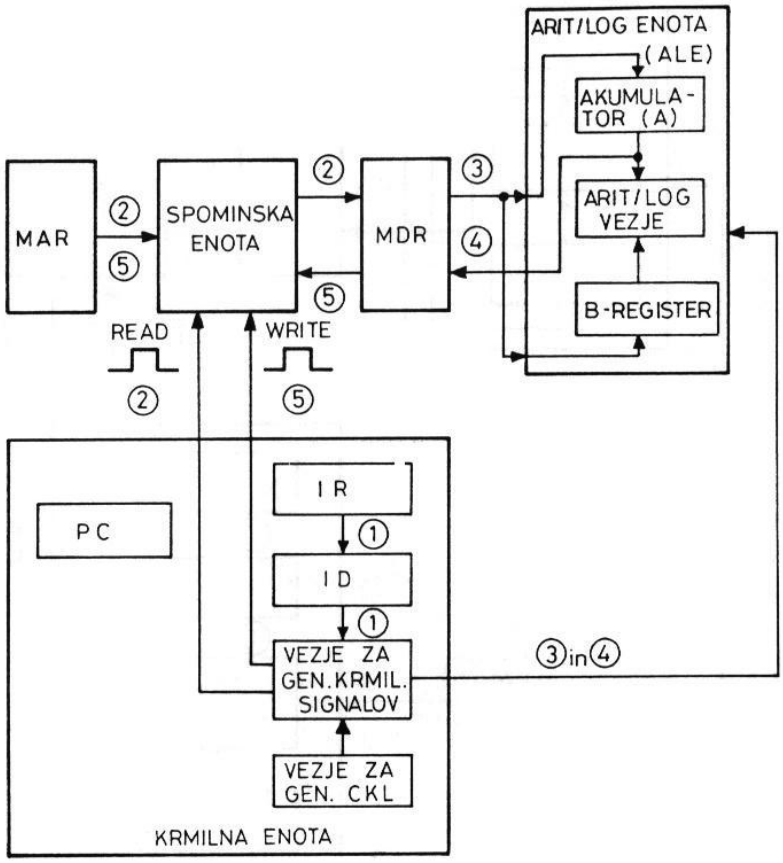
Slika 1.36: Pretok informacij med izvajanjem instrukcijskega cikla

1. Instrukcijski cikel se prične, ko se naslov naslednje instrukcije prenese iz programskega števca (PC) v register spominskih naslovov (Memory Address Register – MAR). Programski števec odloča o tem, katera instrukcija je vzeta iz spomina. Po vsakem instrukcijskem ciklu se njegova vrednost poveča za 1. MAR je uporabljen zato, da shrani le tisti naslov, ki je v tem času dostopen za bralno ali pisalno operacijo.
2. Krmilna enota generira bralni impulz (READ) kar povzroči, da se v spominski enoti prebere vsebina spominske lokacije, ki jo določa MAR. Instrukcijska beseda na tej spominski lokaciji se prenese v register spominskih besed (Memory Data Register – MDR). MDR deluje kot »buffer« register za vse podatke, ki se vpisujejo v ali pa berejo iz spomina.
3. Operacijska koda instrukcijske besede, ki je sedaj v MDR, se prenese v instrukcijski register (IR) v krmilni enoti. Istočasno se prenese naslov operanda instrukcijske besede v MAR (nadomesti njegovo prejšnjo vsebino).
4. Operacijska koda v IR se prenese v instrukcijski dekoder (ID), kjer se operacijska koda »dešifrira«. Ta informacija je potem poslana v vezje za generiranje krmilnih signalov. Tu se formirajo potrebni krmilni signali za izvršitev krmilne instrukcije med izvršilnim (izvajalnim) ciklom.
5. Vsebina programskega števca se poveča za 1, ki omogoča izvajanje naslednjega cikla.

Povzetek instrukcijskega cikla: Med instrukcijskim ciklom je iz spomina vzeta instrukcijska beseda, operacijska koda se dekodira, naslov pa se pošlje v MAR. Stanje PC se poveča za 1.

1.7.1 Izvršilni ali izvajalni cikel

Instrukcijskemu ciklu sledi izvajalni cikel, ki ga narekuje predhodno prebrana in dekodirana operacijska koda. Točne sekvence operacij med izvajalnim ciklom so odvisne od predhodne (trenutno aktualne) instrukcije. Na sliki 1.37 je podan potek informacij tekom izvajalnega cikla.



Slika 1.37 Potek informacij med izvajalnim ciklom mikroračunalnika

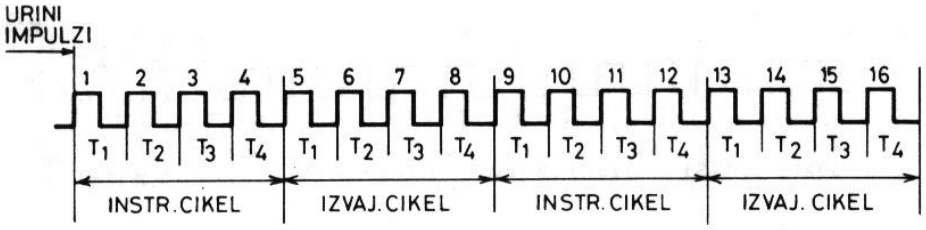
1. IR vsebuje operacijsko kodo in ID (instrukcijski dekoder) določa katera instrukcija naj se izvaja. Tako določi katere krmilne signale naj postavi vezje za generiranje krmilnih signalov v krmilni enoti.
2. Če instrukcija zahteva branje podatkovne besede iz spomina, krmilna enota generira bralni (READ) signal, ki vzame podatkovno besedo z naslova v spominu, ki ga določa MAR in jo pošlje v MDR. Spomnimo se, da vsebuje MAR naslov operanda iz instrukcijskega cikla.
3. Ko je podatek v MDR, je lahko prenešen v ALE, kjer mora biti postavljen v akumulator ali v register B. Krmilna enota generira krmilne signale za izvedbo tega prenosa kakor tudi za odločitev, katera operacija naj se izvaja s podatkovno besedo v ALE.

4. Če instrukcija zahteva prenos podatka v spomin, kot npr. STA (Store Accumulator), se podatek prenese najprej v MDR. Ta podatek mnogokrat pride iz akumulatorja, prenos pa je inicializiran s signalom krmilne enote.
5. Ko je podatek v MDR, krmilna enota generira pisalni impulz (WRITE), ki povzroči, da je podatek vpisan na naslov v spominu, ki ga določa MAR.

Razvidno je, da izvajalni cikel lahko sledi korakom 1, 2 in 3 ali 1, 4 in 5, odvisno od instrukcije, ki se bo izvajala. Ko je izvajalni cikel končan, mikroračunalnik brezpogojno preide zopet v stanje instrukcijskega cikla in vzame naslednjo instrukcijo z naslova, ki ga določa programski števec. Spomnimo se, da je bil programski števec inkrementiran za 1 že med prejšnjim instrukcijskim ciklom.

1.8 Sinhronizacija in krmiljenje mikroračunalnika

Logična vezja, ki sinhronizirajo in krmilijo vse mikroračunalniške operacije, se nahajajo v krmilni enoti. Ta vezja vedno vsebujejo enega ali več urinih generatorjev, ki generirajo periodične urine signale za delovanje kakor tudi časovno zaporedje vseh operacij. Npr. kompletni instrukcijski cikel ali en kompletni cikel izvajanja zahteva 4 urine periode. Na sliki 1.38 je prikazan časovni potek za tak primer.

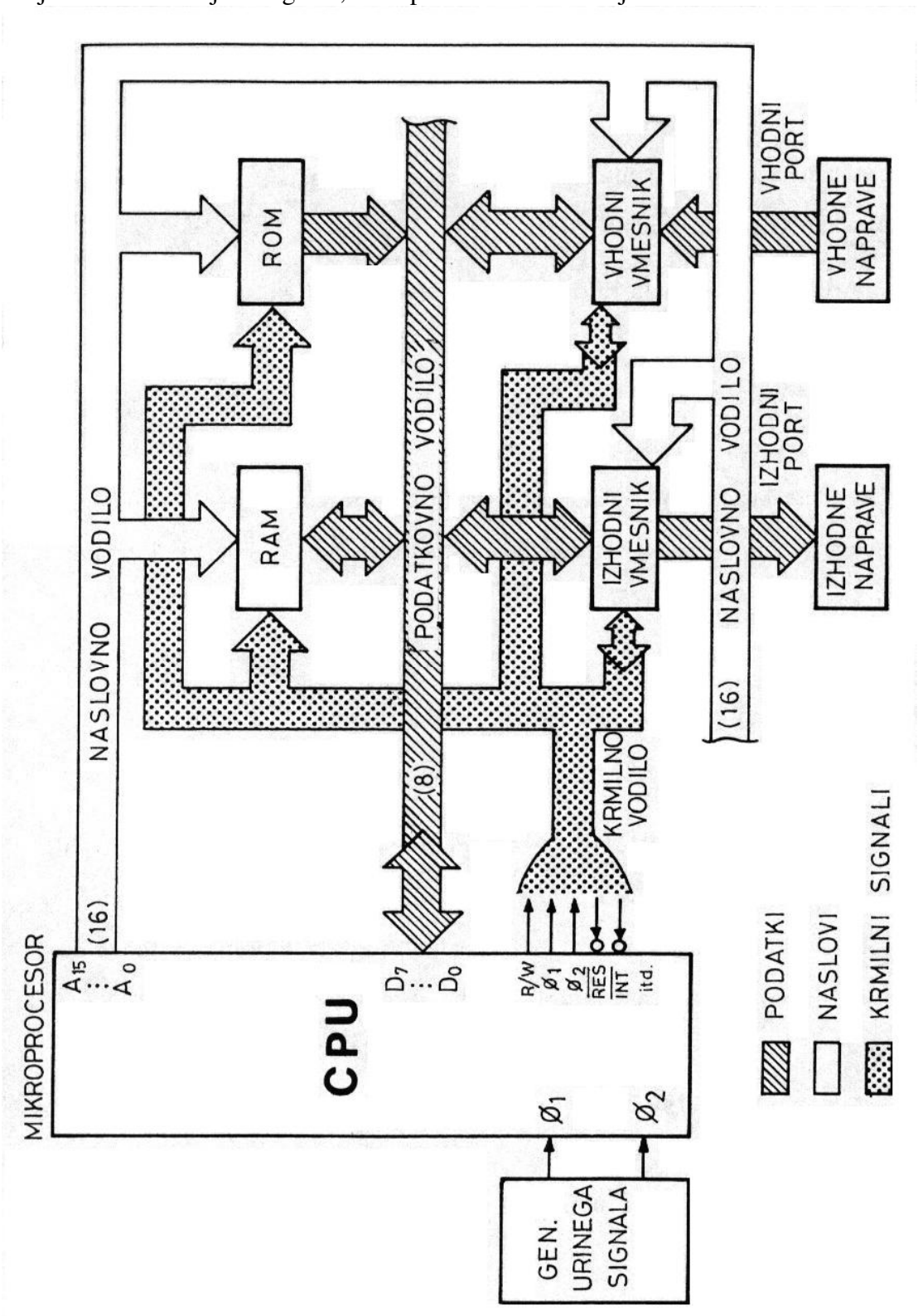


Slika 1.38 Tipična časovna sekvenca za izvajanje mikroračnalniških ciklov

Kot vidimo iz časovnega poteka urinih impulzov na sliki 1.38, vsaka serija štirih urinih period obsega cikel ene operacije. Impulzi od 1 do 6 tvorijo prvi instrukcijski cikel, impulzi 5 do 8 pa tvorijo prvi izvajalni cikel. Impulzi od 9 do 12 tvorijo drugi instrukcijski cikel, impulzi od 13 do 16 pa obsegajo drugi izvajalni cikel. Te sekvence se izvajajo toliko časa, dokler se ne izvedejo vse instrukcije programa.
Vsak operacijski cikel je razdeljen na štiri časovne intervale in določa vse operacije, ki se zgodijo v tem času. Npr. v časovnem intervalu T_1 vsakega instrukcijskega cikla je vsebina PC prenešena v MAR. Ta prenos se izvrši med naraščajočo strmino časovnega cikla T_1.
Npr. med izvajanjem instrukcije ADD pa se seštevanje števil v akumulatorju in v registru B opravi v časovnem intervalu T_4 izvajalnega cikla.
Krmilna enota pošlje krmilne signale k ostalim mikroračunalniškim enotam med vsakim urinim ciklom. Sekvence krmilnih signalov so enake za vse instrukcijske cikle, le sekvence se razlikujejo med ciklom izvajanja odvisno od instrukcije, ki je dejansko v teku.

1.9 Sistem vodil pri mikroračunalniku

Diagram na sliki 1.39 kaže osnovne elemente 8-bitnega mikroračunalniškega sistema in različna vodila, ki jih med seboj povezujejo. Mikroračunalniški sistem ima tri vodila, ki prenašajo vse informacije in signale, ki so potrebni za delovanje sistema.



Slika 1.39: Sistem vodil pri 8-bitnem mikroračunalniku

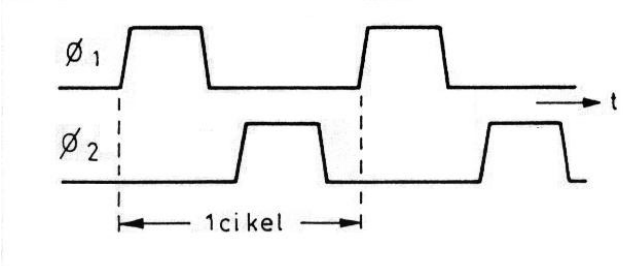
Ta vodila povezujejo mikroračunalnik s spominskimi in vhodno/izhodnimi (Input/Output – vhodno/izhodne) tako, da podatki lahko potujejo med vsemi elementi v sistemu. To pomeni, da je CPE stalno vključena v pošiljanje in v sprejemanje informacij k ali iz spominskih lokacij, izhodnih enot in vhodnih enot.

Nekateri mikroračunalniški sistemi omogočajo neposredno komuniciranje s spominom brez posredovanja CPE. Ta tip delovanja imenujemo direktni (neposredni) dostop do spomina (Direct Memory Acces – DMA) in si ga bomo podrobneje ogledali kasneje.

Na splošno so vsi prenosi informacij povezani s CPE. Če CPE pošilja podatke k drugemu mikroračunalniškemu elementu, se to imenuje pisalna (WRITE) operacija in CPE »piše« v izbrani element. Če CPE sprejema podatke iz nekega elementa, se to imenuje bralna (READ) operacija in CPE bere iz izbranega elementa.
Vodila, ki so vključena v vse prenose podatkov, delujejo na naslednji način:

1. **Naslovno vodilo (Address Bus).** To je enosmerno vodilo, ker se po njem pretakajo informacije le od CPE v spomin in v vhodno/izhodne enote. CPE lahko postavlja logične nivoje na 16-tih linijah naslovnega vodila, s čimer generira $2^{16} = 65536$ različnih možnih naslovov. Vsak od naslovov omogoča dostop do ene spominske lokacije ali vhodno/izhodne enote. Npr. naslov $20A0_{16}$ je lahko lokacija v RAM-u ali ROM-u, kjer je shranjena ena 8-bitna beseda, ali pa je lahko nek 8-bitni »buffer« register, ki je del vmesniškega vezja za tastaturo vhodne naprave. Če CPE želi komunicirati z neko spominsko lokacijo ali vhodno/izhodno enoto, pošlje na svoj 16-bitni naslovni izhod ustrezno naslovno kodo. Ti naslovni biti se potem dekodirajo in tako je omogočen dostop do želene lokacije. Dekodiranje običajno opravi posebno dekodirno vezje, ki ga bomo opisali kasneje.
2. **Podatkovno vodilo**, je dvosmerno, saj lahko podatki potujejo k ali od CPE. Osem priključnih sponk na CPE, D0 do D7, so lahko vhodi ali izhodi odvisno od tega, katero operacijo izvaja CPE (»Read« ali »Write«). Med »Read« operacijo so D0 do D7 v funkciji vhodov in CPU sprejema podatke, ki so bili poslani na podatkovno vodilo iz spomina ali iz vhodno/izhodnih enot, kar je bilo pač izbrano z naslovnim vodilom. Med operacijo »Write« so podatkovni priključki CPU D0 do D7 aktivni kot izhodi in CPE pošilja podatke preko podatkovnega vodila v tisto lokacijo, ki je bila izbrana z naslovno kodo na naslovnem vodilu. V vseh primerih je prenešena podatkovna beseda dolga 8 bitov. V nekaterih mikroračunalnikih so podatkovni priključki uporabljeni tudi za prenos drugih informacij (naslovni biti ali CPE statusne informacije). To pomeni, da so podatkovni priključki multipleksirani, kar pomeni, da mora CPE generirati posebne signale, ki povedo drugim elementom natančno kaj je na podatkovnem vodilu v določenem trenutku.
3. **Krmilno vodilo (Control Bus)** je skupek signalov, ki se uporabljajo za sinhronizacijo posameznih mikroračunalniških elementov. Nekateri od teh krmilnih signalov, kot npr. $R/\overline{W}$ so signali, ki jih pošilja CPE drugim elementom, da jim sporoči, kateri tip operacije se izvaja. Vhodno/izhodne enote lahko pošiljajo krmilne signale k CPE. Najbolj pomemben del krmilnega vodila je sistem urinih signalov, ki generirajo časovne intervale v katerih se vršijo posamezni koraki pri delovanju celotnega sistema. Njihova izvedba se razlikuje od proizvajalca do proizvajalca. Nekateri ne potrebujejo zunanjega vezja za generiranje urinega signala. Na ustrezne sponke je treba priključiti le kvarčni oscilator ali RC vezje. Včasih proizvajalci predvidijo tudi posebna integrirana vezja, ki vsebujejo generatorje urinih signalov. Mnogi uporabljajo sistem dveh urinih signalov z neprekrivajočimi impulzi kot npr. signal na sliki 1.40. Drugi spet delujejo le z enim urinim signalom. V naših razlagah bomo uporabili

sistem z dvema urinima signaloma, ki sta del krmilnega vodila. Nekateri signali so lahko izpeljani iz omenjenih osnovnih dveh in tudi tvorijo del krmilnega vodila.



Slika 1.40: Mikroračunalnik z dvema urinima signaloma

4. **Vhodi in izhodi (Input/Output – I/O).** Med izvajanjem programa CPE nenehno piše v spomin ali pa iz njega bere informacije. Program mora določiti ali naj CPE bere (»Read«) iz neke vhodne enote ali pa naj pošilja podatek v neko izhodno enoto (»Write«). Čeprav naša shema 8-bitnega mikroračunalnika (Slika 1.39) kaže samo eno vhodno in eno izhodno enoto, je lahko na mikroračunalniška vodila priključenih več takšnih enot. Vsaka vhodno/izhodna enota je priključena na sistem vodil preko določenega tipa vmesnika. Vloga vmesnika je, da so podatki med različnimi enotami in mikroračunalnikom prilagojeni. Vmesniki so torej potrebni, če vhodno/izhodne enote uporabljajo drugačne nivoje signalov, časovno zaporedje signalov ali format signalov kot pa mikroračunalnik.

Ko smo omenjali naslovno vodilo, smo ugotovili, da za določitev katerekoli spominske lokacije ali vhodno/izhodne enote, CPE postavi na codilo 16-bitni naslov. To pomeni, da ima vsaka vhodno/izhodna enota točno določen naslov tako kot vsaka lokacija v spominu. V mnogih izvedbah mikroračunalnikov CPE ne loči med spominom in vhodno/izhodnimi enotami in z vsemi komunicira na enak način ob uporabi enakih krmilnih signalov. Ta metoda se imenuje »Memory mapped«. Drugi uporabljajo drugačne krmilne signale in drugačne naslovne dekoderje za vhodno/izhodne enote. To se imenuje izoliran vhodno/izhodni princip. Mi se bomo pri naši razlagi omejili na »Memory Mapped« vhodno/izhodno tehniko, ker je bolj splošna in ima mnoge prednosti pred izolirano vhodno/izhodno tehniko.

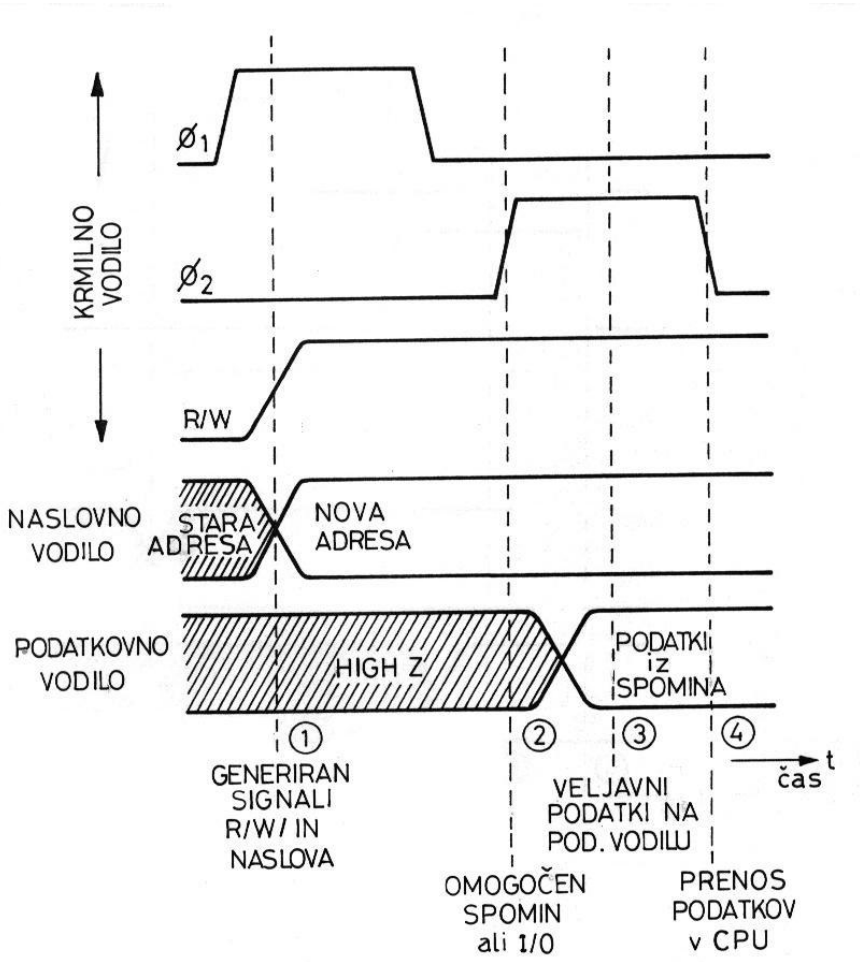
1.10 Bralne in pisalne operacije

Mikroprocesor vsebuje vsa krmilna in aritmetično-logična vezja, ki jih rabi za izvrševanje instrukcij oz. programa, ki je shranjen v RAM in ROM spominu. CPE nenehno opravlja med izvajanjem program bralne in pisalne operacije. Vsako instrukcijo dobimo iz spomina z »READ« operacijo. Glede na interpretacijo instrukcije mora izvršiti »READ« operacijo, da dobi operand iz spomina, ali »WRITE« operacijo, da vpiše podatke v spomin. V nekaterih primerih pa instrukcija zahteva od CPE, da bere podatke iz neke vhodne enote (A/D pretvornik, tastatura,...) ali, da vpiše podatke v neko izhodno enoto (D/A pretvornik, prikazovalnik, itd).

1.10.1 Bralna (READ) operacija

Poglejmo podrobneje, kako se odvija READ operacija:

1. CPE generira pripadajoč logični nivo na $R/\overline{W}$ liniji za inicializacijo »READ« operacije. Pri logičnem stanju $R/\overline{W} = 1$ je v teku bralna operacija. $R/\overline{W}$ linija je del krmilnega vodila in je speljana k vsem spominskim in vhodno/izhodnim enotam.
2. Istočasno postavi CPE na naslovno vodilo 16-bitno naslovno kodo in s tem določi spominsko lokacijo ali Vhodno/Izhodno enoto od koder CPE želi brati podatke.
3. Izbrana spominska ali Vhodno/Izhodna enota postavi 8-bitno besedo na podatkovno vodilo. Vse neizbrane spominske lokacije in vhodno/izhodni elementi ne morejo vplivati na podatkovno vodilo, ker so njihovi tristanjski izhodi v visokoimpedančnem stanju (High Z).
4. CPE sprejme 8-bitno besedo s podatkovnega vodila na podatkovne priključke D_0 do D_7. Ti so v funkciji vhodov, če je $R/\overline{W} = 1$. Ta 8-bitna beseda se potem posreduje v enega od registrov CPE (npr. akumulator). Ta sekvenca je lažje razumljiva s pomočjo časovnega diagrama, ki kaže trenutne medsebojne odnose signalov na različnih vodilih (Slika 1.41).



Slika 1.41: Potek signalov med bralno operacijo

Vse je pogojeno z urinima signaloma ϕ_1 in ϕ_2. Celotna bralna operacija se izvede v enem urinem ciklu. Vodilna strmina ϕ_1 povzroči, da CPE generira $R/\overline{W}$ in naslovne signale. Po kratki zakasnitvi (reda ns) zavzame linija $R/\overline{W}$ visoko logično stanje, naslovno vodilo pa drži novo naslovno kodo. (točka 1 v časovnem diagramu). Naslovno vodilo kaže oba možna

prehoda, (z LOW na HIGH in iz HIGH na LOW), ker se bodo nekatere od 16 naslovnih linij spremenile v eni smeri, druge pa v drugi.

Medtem, ko je prisoten impulz ϕ_2, se sprosti izbrana spominska celica ali vhodno/izhodna enota (točka 2) in pride do prenosa podatkovne besede na podatkovno vodilo. Pri tem je podatkovno vodilo v visokoimpedančnem stanju, ker nobeno nanj priključeno vezje ni bilo sproščeno. Šele pri nastopu ϕ_2 zavzamejo podatki na podatkovnem vodilu končne vrednosti (točka 3). Ponovno je možen prenos podatkov. Zakasnitev med nastopom ϕ_2 in stabiliziranjem podatkov na podatkovnem vodilu je odvisna od hitrosti spomina in vhodno/izhodnih elementov. Pri spominskem elementu je to ti. čas dostopa. Ob nastopu padajoče stranice ϕ_2 se podatki prenesejo v CPE (točka 4). To pomeni, da mora biti spomin ali vhodno/izhodna enota sposobna oddati podatke na podatkovno vodilo predno pride do padajoče strmine ϕ_2, sicer se pripravljeni prenos v CPE ne zgodi. Treba je zagotoviti, da imajo te enote hitrost, ki je kompatibilna z urinim ciklom mikroračunalnika.

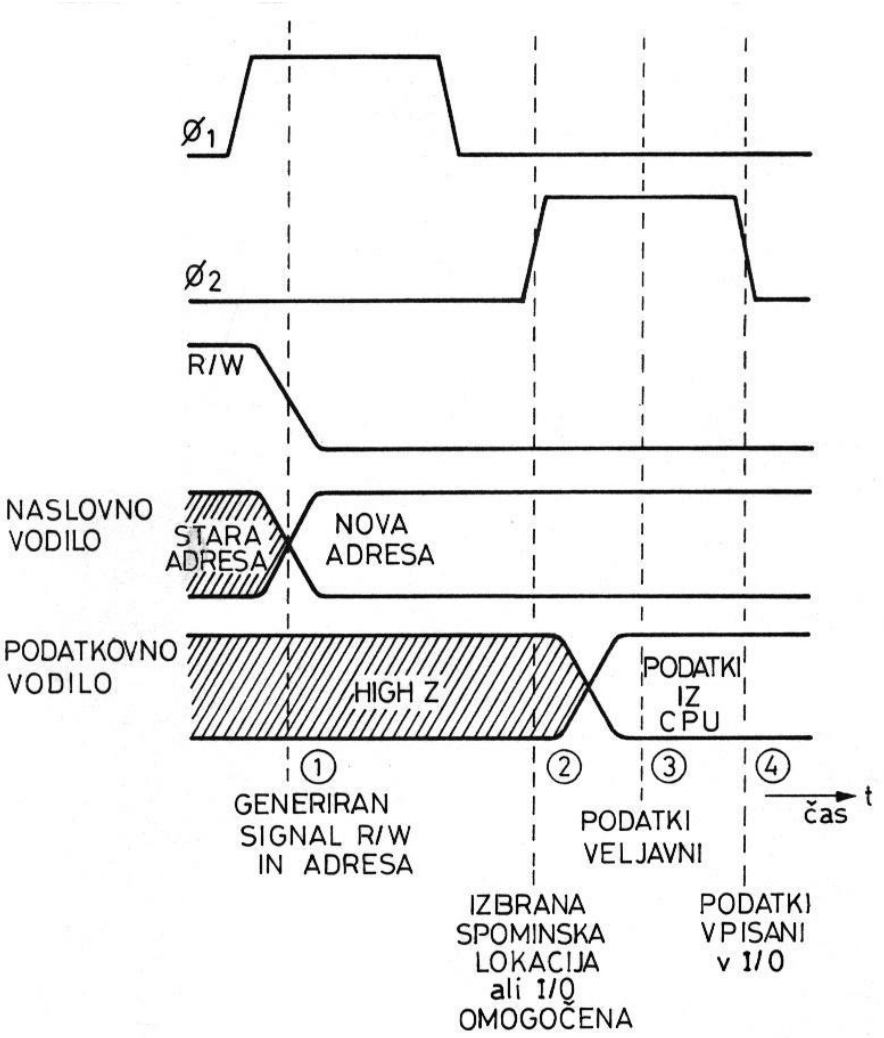
Primer: Imamo PROM, ki ima podatek o času dostopa 750 ns (maksimalno 1μs). Ali ga lahko uporabimo z mikroprocesorjem, ki deluje s frekvenco 1 MHz.

Odgovor: Ne. Pri frekvenci 1 MHz je trajanje impulza ϕ_2 krajše od 500 ns. PROM bi moral imeti čas dostopa krajši od 500 ns za pripravo podatkov za prenos v CPE.

1.10.2 Pisalna (WRITE) operacija

Pisalna operacija poteka v naslednjih korakih:

1. CPE generira logično nivo na liniji $R/\overline{W} = 0$.
2. Istočasno postavi na naslovno vodilo 16-bitno naslovno kodo.
3. CPE odda preko podatkovnih priključkov D0 – D7 8-bitno podatkovno besedo, ki pride iz nekega internega registra (npr. akumulatorja). Vsa ostala vezja, ki so priključena na podatkovno linijo, imajo svoje izhode onesposobljene.
4. Izbrana spominska celica ali vhodno/izhodna enota sprejme podatke s podatkovnega vodila. Vse neizbrane spominske celice in vhodno/izhodne enote nimajo omogočenih vhodov.



Slika 1.42: Potek signalov med pisalno operacijo

Ta sekvenca ima časovni diagram podan na sliki 1.42. Vodilna stranica ϕ_1 inicializira R/$\overline{W}$ signale na naslovnem vodilu (točka 1). Pri nastopu ϕ_2 je sproščena izbrana celica v spominu ali vhodno/izhodni enoti (točka 2) in CPE pošlje podatke na podatkovno vodilo. Nivoji na podatkovnem vodilu se formirajo kratek čas po nastopu impulza ϕ_2. Ti podatki se potem vpišejo v izbrano lokacijo v spominu, ko je ϕ_2 v visokem logičnem stanju. Če je izbrana neka lokacija v vhodno/izhodni enoti, se vanjo vpišejo podatki s podatkovnega vodila med padajočo stranico ϕ_2 (točka 4).

READ in WRITE operacije so glavna zunanja aktivnost CPE. To kaže naslednji primer:

V tabeli 1.19 je kratek program, ki je shranjen v spominskih lokacijah 0020_{16} do 0029_{16} nekega 8-bitnega mikroračunalnika. Ker je vsaka beseda dolga 1 bajt zavzame 16-bitna naslovna koda operanda dva zaporedna bajta.

Tabela: 1.19: Primer programa

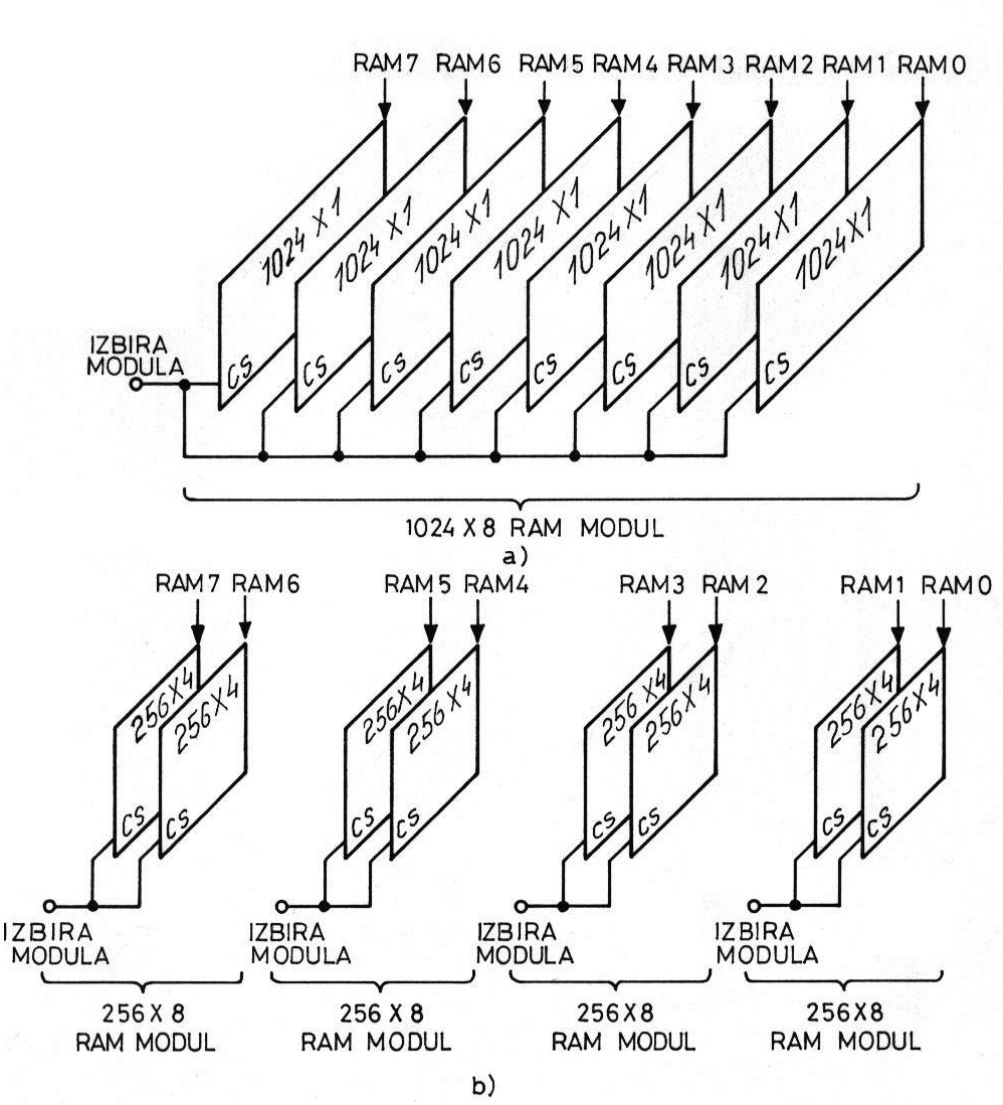
Naslov v spominu (heks.)	Spominska beseda	Simbolična koda	Opis
0020	49	LDA	LOAD akumulator (ACC) z X
0021	01		Naslov operanda X
0022	50		
0023	7E	ADD	ADD Y k vsebini (ACC)
0024	01		Naslov operanda Y
0025	51		
0026	B2	STA	Shrani vsebino ACC
0027	01		Naslov, kjer naj se shrani vsebina akumulatorja
0028	52		
0029	EF	STOP	Program se ustavi

Rešitev:
CPE začne izvajanje programa z branjem vsebine v spominski lokaciji 0020. Beseda, ki je shranjena (49), se prenese v CPE in se interpretira kot instrukcijska koda (z drugimi besedami, CPE vedno interpretira prvo besedo programa kot instrukcijo in programer se mora tega formata vedno držati). CPE dekodira to operacijsko kodo, s tem določi katera operacija se mora izvajati in ugotovi kateri naslov operanda sledi operacijski kodi. V tem primeru je naslov operanda shranjen v dveh zaporednih bajtih v spominu. CPE mora brati lokaciji 0021 in 0022, da dobi naslov podatka X. Ko je naslov prebran, se nadaljuje izvajanje instrukcije z branjem spominske lokacije 0150 in prenese njeno vsebino v akumulator. Celotno izvajanje prve instrukcije zahteva 4 bralne operacije: eno za operacijsko kodo, dve za naslov in eno za operacijo LDA.
Štiri bralne operacije so potrebne tudi za drugo instrukcijo, ki se začne na lokaciji 0023. Ta instrukcija ima prav tako dvobajtni naslov operanda in zahteva prištevanje operanda k vsebini akumulatorja.
Tetja instrukcija se začne na lokaciji 0026 in ima dvobajtni naslov operanda. CPE potrebuje tri READ operacije, da dobi ta naslov. Instrukcija STA pa velja za pisalno operacijo s čimer CPE prenese vsebino akumulatorja v spominsko lokacijo 0152.
Zadnja instrukcija 0029 je enostavno neka operacijska koda brez naslova operanda. CPE prebere to operacijsko kodo (EF), jo dekodira in potem ustavi izvajanje progama.
Skupaj je treba za izvedbo omenjenega programa izvesti 12 bralnih operacij in 1 pisalno operacijo.

1.11 Tehnika naslavljanja

Kot smo že omenili, potrebujemo različna dekodirna vezja, ki omogočajo CPE izbirati spominske lokacije ali vhodno/izhodne enote iz katerih želi brati ali pa v njih vpisovati podatke. Če CPE postavi 16-bitni naslov na vodilo, želimo sprostiti samo eno lokacijo v spominu ali v vhodno/izhodni enoti. Izvedba dekodirnega vezja je odvisna od vrste spominskih vezij, ki jih uporabljamo za RAM ali ROM. Naloga dekodirnega vezja je, da sprosti signal za dostop do izbranega vezja (CHIP SELECT). Tako je aktivno le določeno spominsko vezje ali pa skupina spominskih besed, ki hrani želeno besedo. Preden začnemo z analizo dekodirnih vezij, si poglejmo eno od možnih organizacij spomina.
ROM spomin nekega 8-bitnega mikroračunalnika je izveden z enostavnimi integriranimi vezji, ki hranijo določeno število 8-bitnih besed. Npr. mikroračunalnik mora obsegati 1024

besed oz. (1k) ROM spomina. Izveden je lahko z enim ROM-om kapacitete 1024 x 8 ali pa iz štirih ROM-ov s kapaciteto 256 x 8.
Na drugi strani pa RAM zahteva več logike na vsakem integriranem vezju saj mora biti vanj možno pisati kakor tudi iz njega brati. Tako jih lahko sestavimo na več načinov, da dobimo 8-bitne besede. Slika 1.43 kaže dve splošni metodi za izvedbo 1k RAM-a.

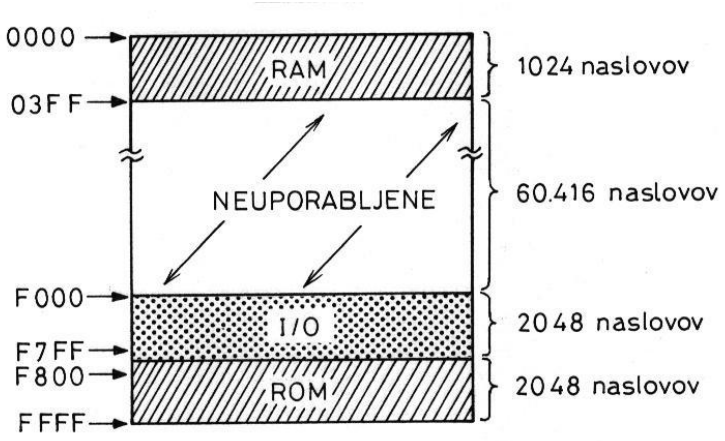


Slika 1.43: Dve izvedbi RAM spomina

Na sliki 1.43a je uporabljenih 8 integriranih vezij s kapaciteto 1024 x 1 RAM, pri čemer vsako integrirano vezje hrani 1 bit besede. Npr. RAM 7 vsebuje najpomembnejše bite (MSB), RAM 0 pa najmanj pomembne bite vseh 1024 besed. Celoten sestav imenujemo tudi spominski modul. V tem primeru imamo 1024-besedni modul. V modulu so vsa integrirana vezja izbrana istočasno, kar je razvidno iz skupne izbirne linije CS (Chip Select).
Na sliki 1.43b imamo uporabljene enote s kapaciteto 256 x 4. Po dve integrirani vezji sestavljata modul 256 x 8. Štirje moduli tvorijo 1k RAM spomina in so sestavljeni iz osmih integriranih vezij 256 x 4. Vsak modul ima svojo izbirno linijo (Modul Select) in samo en modul je izbran v enem časovnem intervalu za branje oz. pisanje.

1.11.1 Spominski prostor

Število naslovov, ki jih pri 16-bitni naslovni kodi lahko CPE izbira, je 65536. Heksadecimalno je obseg naslovov od 0000 do FFFF in tvori celoten spominski prostor. V tipičnem mikroračunalniku je spominski prostor različno izkoriščen. Primer razdelitve spominskega prostora med RAM, ROM in vhodno/izhodne enote je podan na sliki 1.44.

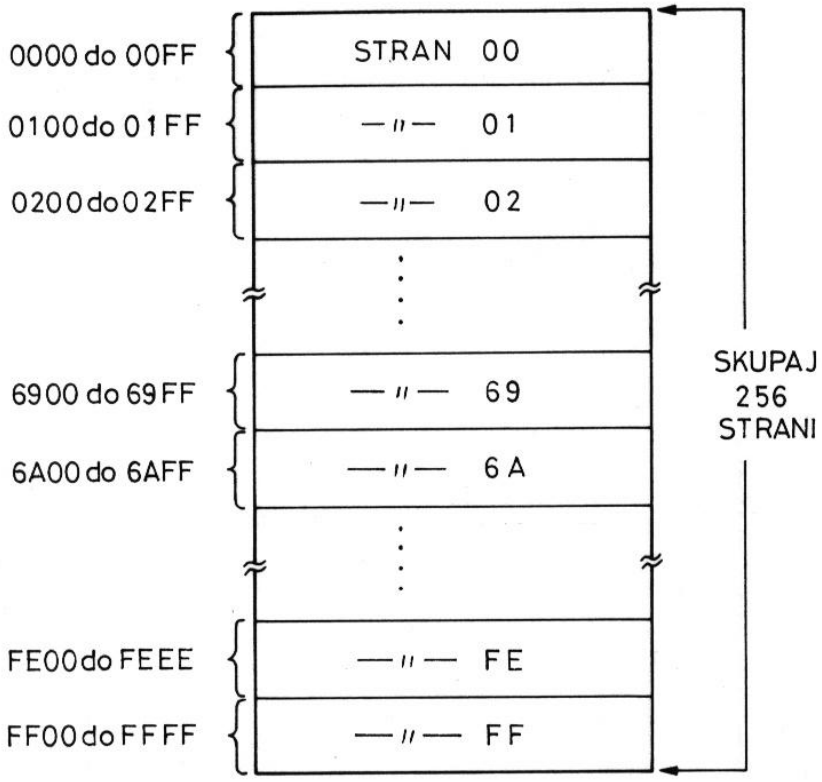


Slika 1.44: Primer razdelitve spominskega prostora

Heksadecimalni naslovi od 0000 do 03FF so rezervirani za RAM. To je skupaj 1024_{10} naslovov. Naslovi od F000 do F7FF so rezervirani za vhodno/izhodne enote, to je skupaj 2048 naslovov. To ne pomeni, da je uporabljenih 2048 vhodno/izhodnih enot za mikroračunalnik, vendar je zelo prikladno izbrati niz naslovov namesto enega za izbiro neke vhodno/izhodne enote. O tem nekoliko kasneje.
Naslovi od F800 do FFFF so rezervirani za ROM, ki hrani trajne programe in podatke, torej skupaj 2048_{10} lokacij v naslovnem prostoru. Prosti naslovi od 0400 do FFFF v našem primeru niso uporabljeni.
Dejanske naslovne lokacije za RAM, ROM in vhodno/izhodne enote izbere načrtovalec glede na izbor mikroračunalnika.

1.11.2 Spominski bloki

Mnogi mikroračunalniki delijo svojih 65536 razpoložljivih naslovov v 256 blokov po 256 naslovov. Vsak od teh blokov imenujemo stran (PAGE). Na sliki 1.45 je prikazano, kako je lahko mikroračunalniški spominski prostor organiziran v bloke (strani).



Slika 1.45: Organizacija naslovnega prostora po straneh

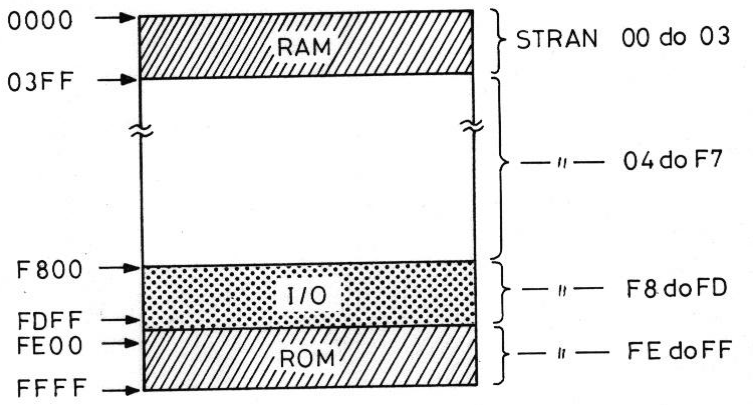
Npr. vsaka stran lahko pokriva 256 naslovov. Tako npr. stran 00 vključuje naslove od 0000 do 00FF, stran 69 vključuje lokacije z naslovi od 6900 do 69FF. Številka strani je torej sestavljena iz prvih dveh heksadecimalnih številk naslova. Tako so vsi naslovi, ki se začnejo s 69, na strani 69.
Če razumemo to organizacijo spomina po straneh, lahko enostavno ločujemo stran in številko besede. Za ilustracijo si poglejmo naslov 1E66:

1E	66
številka strani	beseda na strani

Prvi dve mesti določata stran 1E, zadnja dva pa določata 66. besedo na strani.

1.12 Tehnika naslovnega dekodiranja

Poglejmo si razvoj naslovnega dekodirnega vezja, ki ga rabimo za izbiro ene spominske lokacije ali vhodno/izhodne enote, ki jo naslavlja CPE. Obstaja več pristopov k reševanju dekodirnega problema, ki pa so v osnovi enaki. Namesto, da bi prikazali najrazličnejše izvedbe dekodirne logike, bomo izbrali eno razvojno metodo, katere koncepti in ideje so uporabne pri vseh dekodirnih vezjih. Postopek, ki ga bomo uporabili je neposreden in omogoča blokovno (stransko) organizirano naslavljanje. Razvili bomo celotno dekodirno logiko za 8-bitni mikroračunalnik, ki ima 16-bitne naslove in katerega razdelitev spominskega prostora je prikazana na sliki 1.46.

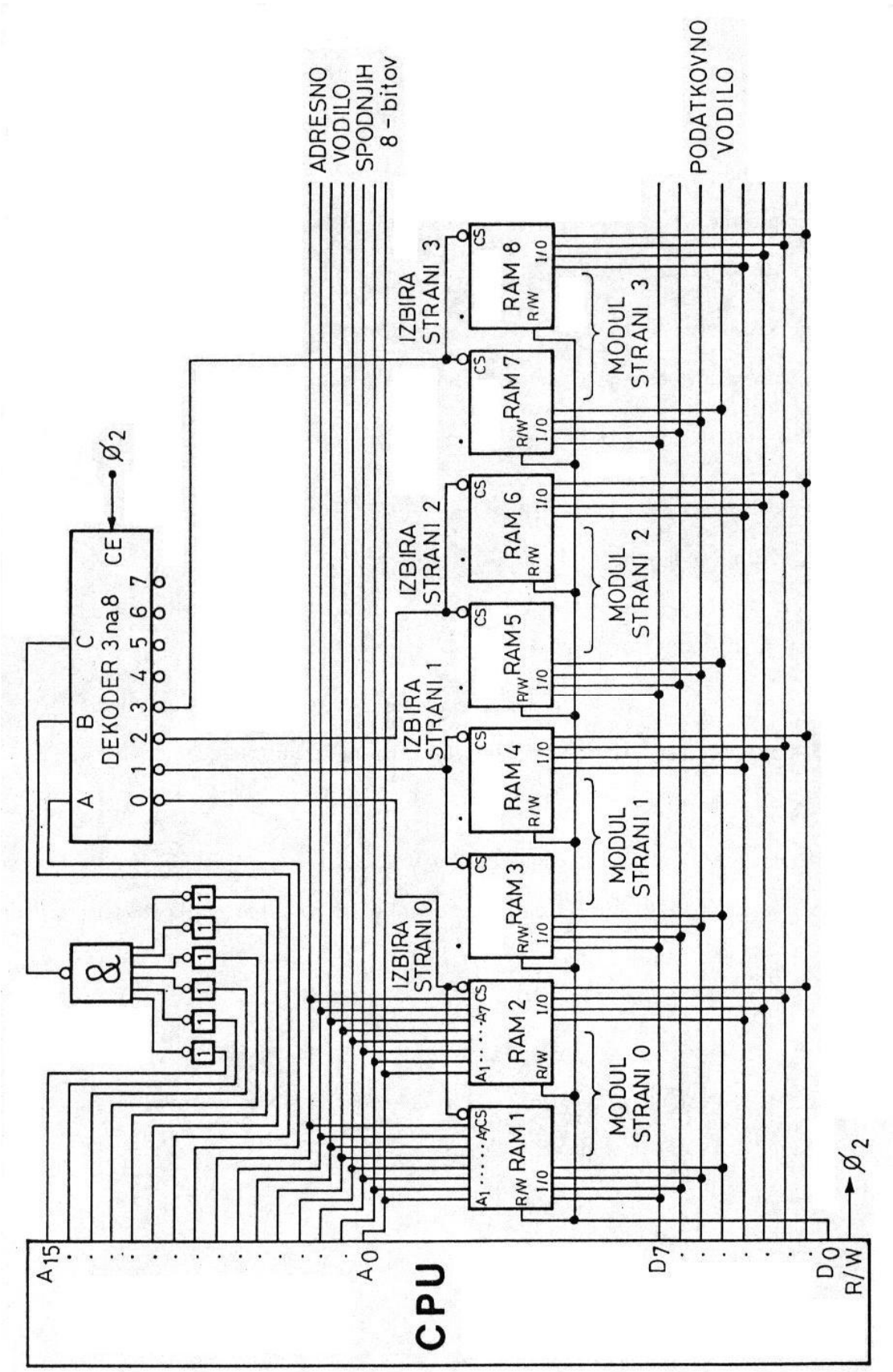


Slika 1.46 Primer razdelitve spominskega prostora

1.12.1 Dekodiranje RAM-a

Preden lahko razvijemo dekodirno logiko za RAM naslove, moramo izbrati vrsto integriranih vezij, ki jih bomo uporabili pri sestavljanju RAM-a. Ker bomo uporabili metodo naslavljanja po straneh, bomo vzeli RAM module na sliki 1.43b, ki lahko shranijo eno stran. Z RAM-ovimi naslovi od 0000 do 03FF rangiranih čez 4 strani, potrebujemo 8 256 x 4 RAM integriranih vezij. Ta vezja imajo 8 naslovnih vhodov, R/$\overline{W}$ vhod, linijo za izbor (Chip Select) in štiri podatkovne vhodno/izhodne linije.
Celotno dekodirno vezje za štiri strani RAM spomina je podano na sliki 1.47. Zapomnite si naslednje podatke:

1. Dve RAM integrirani vezji sestavljata eno stran modula, ki hrani 256 8-bitnih besed. Vsak RAM prispeva 4 bite celotne besede.
2. R/$\overline{W}$ krmilni signal CPE je priključen k R/$\overline{W}$ vhodom vseh RAM integriranih vezij.
3. 8 zgornjih linij naslovnega vodila (A_{15} do A_8) je uporabljenih za dekodirno logiko, ki vključuje dekoder 3 na 8 linij (npr. 74HC138). Ta dekoder vzame tribitno binarno vhodno besedo CBA in aktivira ustrezen izhod za izbiro strani spominskega modula, ki ga želimo naslavljati.
4. 8 spodnjih naslovnih linij CPE, A_7 do A_0, je priključenih k naslovnim vhodom vsakega RAM integriranega vezja. Te naslovne linije izbirajo spominsko besedo iz izbrane strani modula.



Slika 1.47: Dekodiranje RAM-a

Primer:

Vzemimo, da mora CPE izvesti operacijo »WRITE« v spominsko lokacijo $02A5_{16}$. Najprej postavi nizek nivo na $R/\overline{W}$ liniji in generira naslovno kodo na vodilu, ki je v binarni obliki:

$02A5_{16}$	=	0	0	0	0	0	0	1	0	1	0	1	0	0	1	0	1
		A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0

Šest najpomembnejših bitov naslova pogojuje nizek nivo na izhodu NAND vrat, ki je priključen na C vhod dekoderja. Naslovni liniji A_9 in A_8 prenašata logična nivoja 1 oz. 0 na dekoderjeva vhoda B in A. Dekoderjeva binarna vhodna koda je torej CBA = 010, kar pomeni nizek izhodni signal na dekoderjevem izhodu 2. Ta izhodni signal povzroči izbiro strani 2 in sprosti RAM-a 5 in 6. 8 naslovnih linij spodnje vrste izbira naslov 10100101 na strani 2 kamor želi pisati CPE. CPE postavi podatke na podatkovno vodilo in jih vpiše v izbrano lokacijo ob ustreznem signalu ϕ_2. Dekoder ni sproščen pred nastopom ϕ_2, ker slednji sproži njegov sprostitveni vhod CE (Chip Enable).
Treba je torej zagotoviti, da so nivoji na naslovnih linijah stabilni prej kot je spomin aktiviran. Sicer se lahko zgodi, da je aktiviranih več spominskih lokacij, ker naslovne linije spreminjajo stanje.
Primer želi nazorno pokazati kako dekodirno vezje izbere želeni naslov, ki ga posreduje CPE. Poudariti je treba, da smo pri tem izbirali le spominske lokacije od 0000 do 03FF za RAM. Za ta obseg naslovov so vse naslovne linije od A_{15} do A_{10} v nizkem logičnem stanju. Zato dobimo preko NAND vrat in dekodirnika tudi na izhodu C nizko stanje. Naslovne linije A_9 in A_8 dejansko izbirajo strani modula, ki ga naslavljamo. Razvidno je, da bi katerakoli kombinacija na linijah A_{15} do A_{10}, razen samih ničel, povzročila visoko stanje na vhodu C v dekoder. Tako preprečimo, da bi se aktivirali izhodi dekoderja od 0 do 3. Izhodi 4 do 7 niso uporabljeni.

1.12.2 Dekodirno vezje za ROM in vhodno/izhodne enote

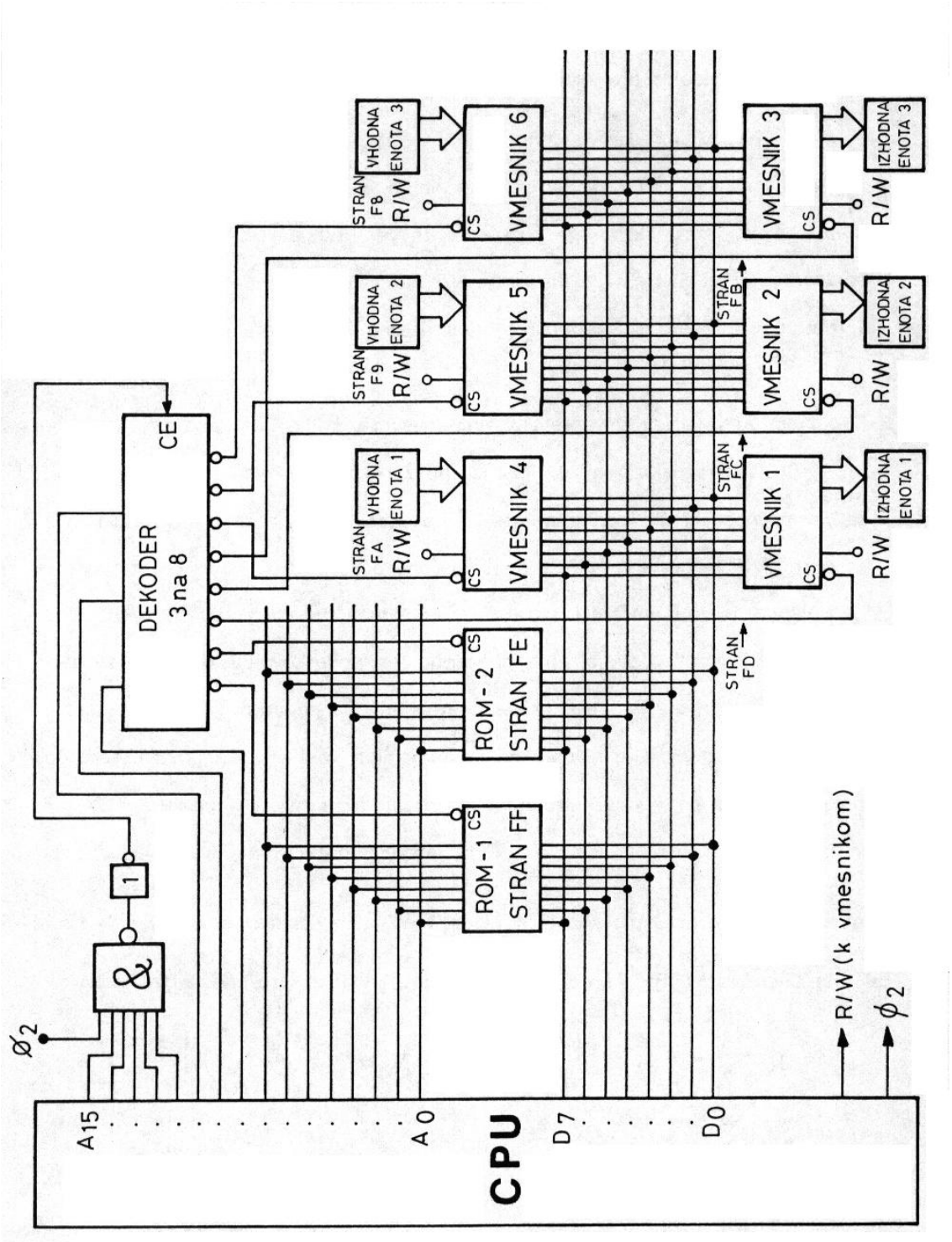
Na sliki 1.48 vidimo, da so naslovi F800 do FDFF (stran F8 do FD) rezervirani za vhodno/izhodne enote, naslovi od FE00 do FFFF (strani FE do FF) pa za ROM. Ker so vse lokacije v istem delu spominskega prostora, lahko razvijamo dekodirno vezje za oba istočasno. Najprej moramo izbrati tip ROM-a, ki ga želimo uporabiti. Pri tem obsegu spomina lahko uporabimo 256 x 8 ROM (npr. 1702 PROM). Za shranjevanje obeh strani rabimo dve integrirani vezji.
Ko analiziramo vhodno/izhodne enote, predpostavimo, da je mikroračunalnik priključen na 6 različnih perifernih enot – tri vhodne in tri izhodne enote. Vsaka enota naj ima po en vmesnik, ki jo povezuje z mikroračunalnikom. Ker smo lokalizirali šest strani spominskega prostora za vhodno/izhodne enote, lahko dodelimo eno stran vsakemu vmesniškemu vezju. Z drugimi besedami: mi lahko dodelimo vse naslove na strani FD vmesniku 1, vse naslove na strani FC vmesniku 2, itd. Smisel tega početja je v tem, da se dekodiranje omeji samo na 8 zgornjih linij naslovnega vodila. V spodnji tabeli je izpisana razporeditev vhodno/izhodnih enot v spominskem prostoru.

Tabela 1.20: Razporeditev vhodno/izhodnih enot v spominskem prostoru

Vmesnik	Vhod ali izhod	Naslov
1	Izhod	FD00 do FDFF (stran FD)
2	Izhod	FC00 do FCFF (stran FC)
3	Izhod	FB00 do FBFF (stran FB)
4	Vhod	FA00 do FAFF (stran FA)
5	Vhod	F900 do F9FF (stran F9)
6	Vhod	F800 do F8FF (stran F8)

Celotno dekodirno vezje za vhodno/izhodne enote in ROM je prikazano na sliki 1.48. Zapomnimo si naslednje točke:

1. Uporabljena so ROM vezja 256 x 8, vsak za eno stran.
2. Vmesniška vezja 1, 2 in 3 so namenjena izhodnim enotam, vmesniška vezja 4, 5 in 6 pa vhodnim enotam. Vsa vmesniška vezja so lahko različna. Lahko so le ojačevalniki (bufferji) ali FF registri ali pa kar celotna vmesniška vezja.
3. 8 zgornjih naslovnih linij CPE uporabimo za izvedbo dekodirne logike, katere osrednji del je dekoder 3 na 8. Ta dekoder izbira izhode za osem strani (Page Select Inputs), od katerih mora biti aktiviran le eden, kar je odvisno od naslovne kode.
4. Osem spodnjih naslovnih linij CPE je priključenih na naslovne vhode vsakega ROM-a za izbiro ene besede z izbrane strani.
5. $R/\overline{W}$ linija CPE ni priključena na posamezna ROM vezja, ker jih ni treba vpisovati. $R/\overline{W}$ pa je priključena na vmesnike. Za to sta dva razloga. Nekaj vmesniških vezij je sposobnih dejanskega vpisovanja ali branja, drugi razlog pa je v tem, da se onemogoči »nesporazume« na podatkovnem vodilu. Za ilustracijo naj bo izbran vmesnik 4 z nekim naslovom na strani FA, ker CPE želi brati podatke iz vhodne enote 1. Predpostavimo, da zaradi pomote v programu, CPE opravlja WRITE operacijo na naslovu FA00. Ta naslov izbere vmesnik 4 in želi postaviti podatke iz vhodne enote na podatkovno vodilo. Če pa bi CPE opravljala pisalno operacijo na podatkovni besedi internega registra, bi bila tudi ta beseda postavljena na podatkovno vodilo. Ta dva različna niza logičnih nivojev na podatkovnem vodilu lahko povzročita težave CPE ali pa vmesniškemu vezju. To težavo rešimo z uporabo $R/\overline{W}$ linije, ki sprosti vmesnik 4 s pisalno operacijo $R/\overline{W} = 0$.



Slika 1.48: Dekodiranje ROM-a in vhodno/izhodnih enot

Primer: Opišite proces, pri katerem dekodirno vezje izbere naslov FE77 v ROM-u.

Rešitev: CPE postavi na naslovno vodilo FE77. Binarno izraženo je ta koda:

$FE77_{16}$	=	1	1	1	1	1	1	1	0	0	1	1	1	0	1	1	1
		A_{15}	A_{14}	A_{13}	A_{12}	A_{11}	A_{10}	A_9	A_8	A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0

Pet najpomembnejših bitov v naslovu (A_{15} do A_{11}) pogojuje nizek nivo na izhodu NAND vrat (Če je $\Phi_2 = 1$), ki je invertiran in priključen na dekoderjev sprostitveni vhod (CE). S tem se aktivira dekoder. Naslovne linije A_{10}, A_9 in A_8 imajo vrednost 110, kar pomeni, da je na izhodu dekoderja aktiven izhod 6. Nizek signal na izhodu 6 aktivira izbirno linijo (CS) za stran FE.
Spodnja vrsta naslovnih linij (A_7 do A_0) izbira lokacijo 0111 0111 v ROM-2. To je naslovljena lokacija, katere beseda se pojavi na podatkovnih linijah drugega ROM-a. Beseda ne pride na podatkovno vodilo pred nastopom Φ_2. Med prisotnostjo Φ_2, CPU sprejme besedo s podatkovnega vodila in jo prenese v enega od njenih internih registrov.

Primer: Opišite proces, pri katerem CPE vpiše besedo v izhodno enoto 2.

Rešitev: Izbirni vhod (CS) vmesnika 2 je prožen z izhodom 4 dekoderja. To je izbirna linija za stran FC. To pomeni, da vsak naslov, ki ima prvi dve mesti heksadicemalne vrednosti naslova enako FC, izbira vmesniško vezje 2 in njegovo izhodno napravo 2. Za vpis podatkov v izhodno enoto 2 mora CPE postaviti signal $R/\overline{W} = 0$ in generirati naslov v obliki FCXX, pri čemer je lahko XX katerokoli dvomestno heksadecimalno število. Beseda, ki jo želimo vpisati, se postavi na podatkovno vodilo, od koder se potem prenese preko vmesniškega vezja na izhod 2.

Ta dva primera kažeta, kako naslovno dekodirno vezje izbira spominske lokacije, ki so določene s CPE, v ROM-u ali v vhodno/izhodnih napravah. Pri oblikovanju tega vezja je najpomembneje, da so pri vseh naslovih v območju od F800 do FFFF naslovne linije A_{15} do A_{11} v visokem logičnem stanju. To pogojuje visok nivo na CE vhodu dekoderja. Naslovne linije A_{10}, A_9 in A_8 potem izbirajo pripadajoč dekoderjev izhod. Pri tem je treba paziti, da vsaka kombinacija na linijah A_{15} do A_{11}, kjer ne bi bili vsi vhodi v visokem logičnem stanju, povzroči nizek logični nivo na CE, kar pomeni, da so vsi izhodi dekoderja v visokem logičnem stanju in noben ROM ali vhodno/izhodna enota ne more biti izbrana.

Celotno dekodirno vezje za vse omenjene primere dobimo s kombinacijo vezij s slik 1.47 in 1.48. Vezje vključuje dvoje NAND vrat, sedem inverterjev, dva dekoderja 3 na 8, osem 256 x 4 RAM-ov, dva 256 x 8 ROM-a in šest različnih vmesniških vezij. Če je treba kasneje povečati prostor za RAM ali ROM, je treba dodati dekodirno vezje. Nekateri proizvajalci vključujejo v svoj naslovni prostor dovolj dekodirnih vezij, da kasnejša razširitev zahteva le dodatna ROM in RAM vezja.

1.13 Mikroprocesor

Mikroprocesor je osrednji sklop mikroračunalnika oz. mikrokrmilnika. Njegova hitrost določa maksimalno hitrost mikroračunalnika, njegovi naslovni in podatkovni priključki pa določajo kapaciteto spomina in velikost (dolžino besed) medtem ko krmilni priključki določajo vrsto vhodno/izhodnih enot, ki jih lahko nanj priključimo.

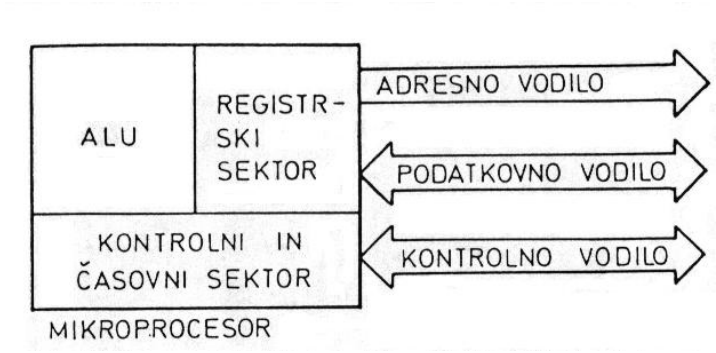
Mikroprocesor opravlja več funkcij:

1. Skrbi za časovne in krmilne signale za vse elemente mikroračunalnika oz mikrokrmilnika.
2. Pobira podatke in instrukcije iz spomina.
3. Prenaša podatke v in iz vhodno/izhodnih enot.
4. Dekodira instrukcije.
5. Vrši aritmetične in logične operacije, ki jih velevajo instrukcije.
6. Odgovarja na krmilne signale, ki jih generirajo vhodno/izhodne enote kot npr. RESET oz. prekinitve (INTERRUPT).

Mikroprocesor vsebuje vsa logična vezja za izvajanje teh funkcij, vendar je treba upoštevati, da velik del njegove notranje logike od zunaj uporabniku ni dostopen. Npr. z zunanjim signalom ne moremo inkrementirati programskega števca (PC). So pa ti elementi dostopni programsko. To pomeni, da lahko vplivamo na notranja vezja samo preko uporabniškega programa.
Zato je dobro poznati notranjo konstrukcijo in značilnosti mikroprocesorja. Mikroprocesorska logika je zelo kompleksna in jo lahko razdelimo v tri bloke ali sektorje:

1. Registrski sektor,
2. Krmilni in časovni sektor,
3. ALE.

Čeprav se ti sektorji med seboj prekrivajo, jih bomo obravnavali ločeno.



Slika 1.49 Glavni funkcijski bloki mikroprocesorja.

1.13.1 Časovni in krmilni sektor

To je edini blok mikroprocesorja, na katerega lahko zelo malo vplivamo in ga pri razvijanju programov ni treba podrobneje poznati. Njegovi glavni funkciji sta branje in dekodiranje instrukcij, ki se nahajajo v programskem spominu ter generiranje krmilnih signalov, ki jih zahteva ALE in registrski sektor za izvajanje instrukcij.

1.13.1.1 Signali krmilnega vodila

Krmilni sektor generira tudi zunanje krmilne signale, ki jih pošilja drugim mikroprocesorskim enotam. Poleg generiranja izhodnih signalov krmilnega vodila, krmilni sektor tudi odgovarja vhodnim signalom, ki jih mikroprocesorju pošiljajo druge enote.

Vsak mikroprocesor ima svoj lasten nabor vhodnih in izhodnih krmilnih signalov, ki so podrobno opisani v priročnikih proizvajalcev. Vseh v tem poglavju ne bomo razlagali. Opisali bomo nekatere pomembnejše krmilne signale, ki so skupni večjemu številu mikroprocesorjev.

RESET
Ko se ta vhod aktivira, se večina mikroprocesorskih registrov postavi na 0. V mnogih mikroprocesorskih vezjih se programski števec resetira tako, da se instrukcija, ki je shranjena na spominski lokaciji 0000_H, izvaja prva. V nekaterih mikroprocesorjih pa aktiviranje RESET vhoda ne briše programskega števca. Namesto tega se ta napolni z vsebino dveh specifičnih spominskih lokacij (vsaka hrani en bajt 16-bitnega naslova). Navadno je ta startni naslov shranjen v ROM-u in je često imenovan naslovni vektor.

$R/\overline{W}$
Ta mikroprocesorska vhodna linija obvešča ostale enote ali je v teku bralna ali pa pisalna operacija. Nekateri uporabljajo ločene krmilne linije, pri čemer RD označuje bralno, WR pa pisalno operacijo.

MREQ (Memory request)
Ta izhod označuje, da je v teku dostop do spomina.

HOLD
Ta vhod se uporablja za operacije neposrednega dostopa do spomina, o katerih bomo govorili kasneje. Ko zunanja enota aktivira ta vhod, mikroprocesor zaključi izvajanje instrukcije in loči svoje naslovno in podatkovno vodilo. To pomeni, da onemogoči tristanjske podatkovne in naslovne priključke, tako da so dejansko izklopljeni. Na ta način lahko druge zunanje enote uporabljajo naslovno in podatkovno vodilo vse dotlej, dokler je HOLD aktiven.

HLDA (Hold Acknowledge)
To je izhodni signal, ki ga generira mikroprocesor, da sporoči zunanji logiki, da je mikroprocesor v stanju HOLD in da sta podatkovno in naslovno vodilo na razpolago. Podoben mikroprocesorski izhod je BA (Buses Available).

INT ali *IRQ* (Interrupt Request – zahteva po prekinitvi)
To je mikroprocesorski vhod, ki ga uporabljajo vhodno-izhodne enote za prekinitev izvajanja tekočega programa in za preskok v poseben program prekinitvene servisne rutine. Mikroprocesor izvaja ta poseben program, ki običajno vključuje servisiranje prekinitvene enote. Ko je to izvajanje zaključeno, mikroprocesor prevzame izvajanje programa, ki ga je obdeloval med prekinitvijo.

INTE (Interrupt Enable – omogočanje prekinitve)
To je mikroprocesorski izhod, ki obvesti zunanje enote ali je interna mikroprocesorska logika za prekinitev omogočena ali ne. Če ni omogočena, ne bo reagiral na spremembo logičnega nivoja na liniji INT ali IRQ. Stanje INTE lahko določimo programsko.

NMI (Nonmaskable Interrupt – nemaskirana prekinitev)
Ta prekinitveni vhod se razlikuje od INT oziroma IRQ po tem, da njegov vpliv ne moremo onemogočiti. To pomeni, da ustrezen signal na NMI vedno povzroči prekinitev, ne glede na stanja, ki omogočajo prekinitve.

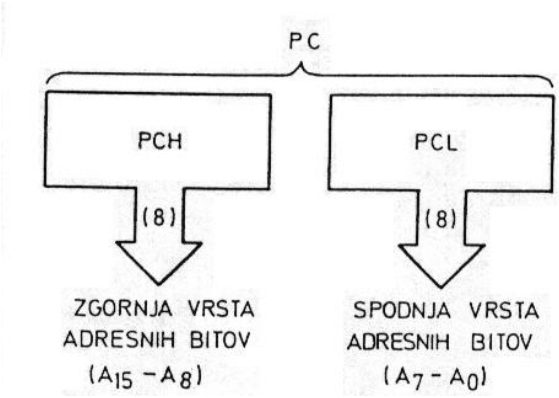
1.13.2 Registrski sektor

Najpogostejša operacija znotraj mikroprocesorja je prenos binarnih podatkov iz enega registra v drugega. Število in mesto registrov je ključni podatek, ki določa učinkovitost programiranja pri reševanju neke naloge. Zgradba registrov pri različnih mikroprocesorjih se od proizvajalca do proizvajalca zelo razlikuje. Vendar pa so osnovne funkcije, ki jih vršijo različni registri v bistvu enake pri vseh mikroprocesorjih. Služijo za shranjevanje podatkov, naslovov instrukcijskih kod na različnih nivojih delovanja mikroprocesorja. Nekateri se uporabljajo kot števci (možno jih je programsko krmiliti) za zasledovanje npr. števila časovnih intervalov, v katerih se posamezne instrukcijske sekvence izvajajo oz. lokacij sekvenčnega spomina od koder je treba vzeti podatke.

Opisali bomo najpogostejše vrste registrov, njihove funkcije in način, kako nanje programsko vplivamo.

INSTRUKCIJSKI REGISTER (IR)
Ko CPE pobere instrukcijsko besedo iz spomina, jo pošlje v IR. Tu se shrani, dokler instrukcijsko dekodirno vezje ne odloči, katera instrukcija naj se izvaja. CPE avtomatično uporablja IR med vsakim instrukcijskim ciklom in programerju ni nikoli treba posegati po tem registru. Dolžina IR je enaka dolžini podatkovne besede. Za 8-bitni mikroprocesor znaša dolžina IR 8 bitov.

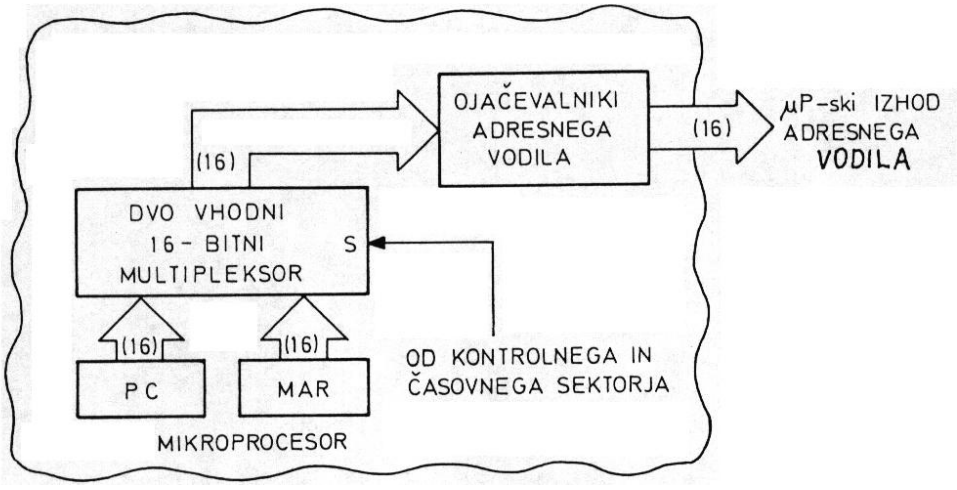
PROGRAMSKI ŠTEVEC (PC – Program Counter)
Programski števec vedno vsebuje v spominu naslov instrukcije, ki naj se naslednja izvaja. Ko se aktivira RESET vhod, se programski števec postavi na naslov prve instrukcije, ki naj se izvaja.
Mikroprocesor postavi vsebino programskega števca na naslovno vodilo in dobi prvi bajt instrukcije iz te spominske lokacije. Mikroprocesor po vsakem koraku inkrementira programski števec in na ta način sekvenčno izvaja sklenjen program, razen če program ne vsebuje kakšne instrukcije, ki spremeni zaporedje izvajanja (npr. JUMP). Dolžina besede, ki je v programskem števcu, je odvisna od števila naslovnih bitov, s katerimi lahko mikroprocesor manipulira. Večina znanih mikroprocesorjev uporablja 16-bitne naslove, nekateri pa tudi 12-bitne. V obeh primerih pa ima programski števec enako število bitov kot naslovno vodilo. Pri mnogih mikroprocesorjih je programski števec razdeljen na dva manjša registra (PCH in PCL), od katerih vsak vsebuje eno polovico naslova. Na naslednji sliki je prikazan 16-bitni programski števec, ki je razdeljen na dve 8-bitni enoti.



Slika 1.50: Delitev 16-bitnega programskega števca na višji in nižji bajt

Smisel uporabe PCH in PCL je v pogosti potrebi po shranjevanju njegove vsebine v spomin. Ker lahko v 8-bitni spomin shranjujemo le 8-bitne besede, mora biti PC razdeljen v dve polovici, ki se shranita v dve zaporedni spominski lokaciji.
Programer nima neposrednega dostopa do PC. To pomeni, da ni namenskih instrukcij za spreminjanje vsebine programskega števca (polnjenje PC iz spomina ali pa shranjevanje vsebine PC v spomin). Vendar obstajajo mnoge instrukcije, ki povzročajo, da vsebina programskega števca zavzame drugačno vrednost kot pa je normalna sekvenca. Ukaz JUMP je primer take instrukcije, ki spremeni vsebino programskega števca tako, da CPE izvaja instrukcijo izven običajnega zaporedja.

MAR (Memory Address Register – Register spominskih naslovov)
Služi za shranjevanje naslovov podatkov, ki jih CPE bere iz spomina, ali pa jih vanj vpisuje. Če se vrši operacija ADD, se del naslova, ki pripada operandu tega ukaza, shrani v MAR. Vsebina MAR je nato prenešena na naslovno vodilo, tako da lahko CPE vzame podatke med naslednjim urinim impulzom.
Kot vidimo, imamo dva vira naslovov za mikroprocesorsko naslovno vodilo in sicer programski števec ter register spominskih naslovov. Za preklop PC ali MAR na naslovno vodilo skrbi multiplekser, glede na to ali je CPE v instrukcijskem ali v izvajalnem ciklu (Slika 1.51).



Slika 1.51: Izbira vira naslova na vodilu

Vidimo, da sta izhoda PC in MAR vezana na multipleksor, katerega izhod je vezan na mikroprocesorjevo naslovno vodilo. Izbirni vhod S multipleksorja je krmiljen s signalom iz krmilnega sektorja.

AKUMULATOR
Akumulator je register, v katerega se prenašajo rezultati skoraj vseh operacij, ki jih vrši ALE. V mnogih instrukcijah ALE je akumulator vir enega od operandov in naslov rezultata.
Poleg teh pa ima akumulator še druge funkcije. Lahko npr. služi kot shranjevalni register za podatke, ki se pošiljajo neki izhodni enoti ali pa kot sprejemni register za podatke, ki se odčitavajo na neki vhodni enoti. Nekateri mikroprocesorji imajo več akumulatorjev. Pri teh mikroprocesorjih določajo instrukcijske kode, kateri akumulator naj se uporabi.

V splošnem ima akumulator enako število bitov kot je dolžina mikroprocesorske besede. Nekateri mikroprocesorji imajo tudi »podaljševalni register«, ki se uporablja skupaj z akumulatorjem za obdelavo binarnih števil z več kot 8 biti. Npr. mikroprocesor omogoča izvajanje »aritmetike z dvojno natančnostjo«, v kateri je vsako število dolgo 16 bitov in je shranjeno v dveh besedah spomina. Ko se te 16-bitne podatkovne besede pošljejo CPE, podaljševalni register shrani 8 najmanj pomembnih bitov, akumulator pa shrani 8 najpomembnejših bitov. Ta dva registra lahko veljata za 16-bitni register.

V mnogih mikroprocesorjih ALE ne more neposredno vršiti neke operacije na besedi v spominu. Besedo je treba najprej prebrati iz spomina ter namestiti v akumulator. Operacija se izvaja na vsebini akumulatorja in se nato shrani nazaj v spomin. Takšna sekvenca zahteva tri instrukcije in je ilustrirana na spodnjem primeru:

Tabela 1.21: Primer programa

	Heksadec. koda	Opis
Instrukcija 1	A9	Load Accumulator (LDA)
	20	Naslov podatka (0520_{16})
	05	
Instrukcija 2	67	Complement Accumulator (CMA)
Instrukcija 3	85	Store Accumulator (STA)
	20	Naslov rezultata (0520_{16})
	05	

Prva instrukcija zahteva tri bajte in omogoči nalaganje akumulatorja z vsebino na spominski lokaciji 0520_{16}. Druga instrukcija obsega le en bajt in tvori eniški komplement vsebine akumulatorja. Tretja instrukcija spet obsega tri bajte in shrani vsebino akumulatorja nazaj na lokacijo 0520_{16} v spominu.
Nekateri mikroprocesorji dopuščajo, da se določene operacije vršijo na spominskih besedah brez uporabe akumulatorja kot vmesne shranjevalne lokacije. To precej zmanjša število instrukcij, zahteva pa kompleksnejše mikroprocesorsko vezje. V ilustracijo bi zgornja operacije izgledala tako, kot je prikazano v tabeli 1.22.

Tabela 1.22: Primer operacije nad podatkovno besedo brez uporabe akumulatorja

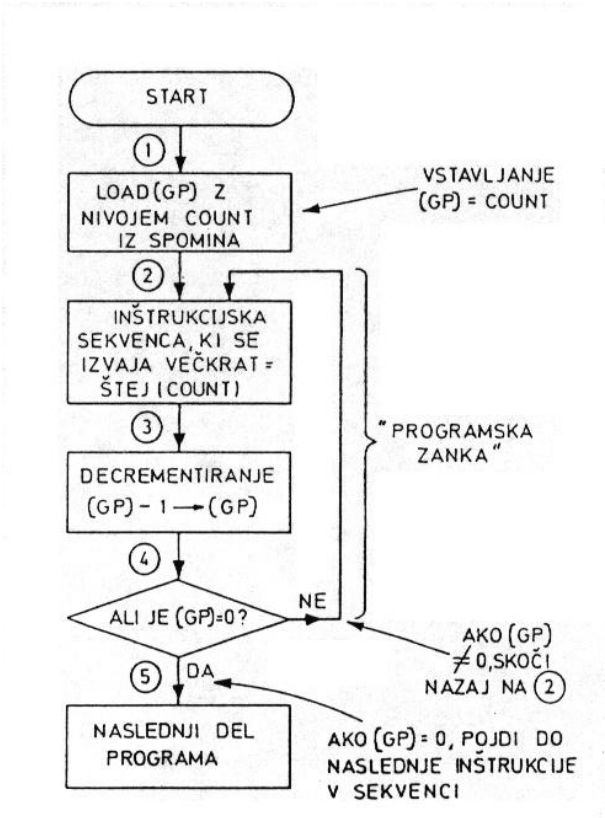
	Heksadec. koda	Opis
Instrukcija 1	69	Complement Memory (CMM)
	20	Naslov podatka (0520_{16})
	05	

VEČNAMENSKI REGISTER (General Purpose Register)
Večnamenski registri se uporabljajo za sranjevanje podatkov, ki jih med izvajanjem programa večkrat potrebujemo. S tem dosežemo hitrejše izvajanje programa, saj CPE ni treba vršiti operacij branja iz spomina vsakokrat, ko potrebuje podatek. Služijo lahko tudi za shranjevanje delnih rezultatov aritmetičnih operacij. V nekaterih mikroprocesorjih se večnamenski registri lahko uporabljajo tudi kot indeksni registri in kot akumulatorji. Število večnamenskih registrov se med posameznimi tipi mikroprocesorjev zelo razlikuje (med 0 in 12).

Obstaja več instrukcij, ki jih lahko programer uporablja za dostop do večnamenskih registrov (GP)
Najpogostejše so:

1. Napolni GP iz spomina - $[M] \rightarrow [GP]$
2. Shrani vsebino GP v spomin - $[GP] \rightarrow [M]$
3. Prenos vsebine GP v akumulator - $[GP] \rightarrow [A]$
4. Prenos vsebine akumulatorja v GP - $[A] \rightarrow [GP]$
5. Prenos vsebine enega GP v drugi GP - $[GP] \rightarrow [GP]$
6. Povečanje vsebine GP (inkrementiranje) - $[GP + 1] \rightarrow [GP]$
7. Zmanjšanje vsebine GP (dekrementiranje) - $[GP - 1] \rightarrow [GP]$

Zadnji dve instrukciji sta za programerja zelo koristni. Omogočata mu uporabo GP kot števca, ki prišteva oz. odšteva cikle, v katerih se posamezna operacija ali zaporedje operacij vrši. To je ilustrirano na sliki 1.52, ki kaže diagram poteka dela programa. Vidimo, kako se GP uporablja za odštevanje ciklov, v katerih se instrukcijska sekvenca v bloku 2 izvaja. Na začetku programa se GP napolni iz spomina z vrednostjo COUNT (blok 1). To je število ponovitev sekvence v bloku 2. Po izvršitvi bloka 2 se vsebina GP zmanjša (blok 3). Program se nato pomakne do odločitvenega bloka 4. Če je vsebina GP različna od nič, se ponavlja izvajanje blokov 2 do 4. Šele, ko je vsebina GP enaka 0, se izvajanje programa premakne v blok 5. S pomočjo takšnega registra, ki ga uporabljamo kot indeksni register, lahko izvajamo programske zanke.



Slika 1.52: Uporaba večnamenskega registra za tvorjenje programskih zank

INDEKSNI REGISTER

Indeksni register se lahko uporablja za splošne shranjevalne funkcije CPE in v funkciji števca. Ima pa še posebno funkcijo, ki je v veliko pomoč pri programu, pri katerem je treba obdelovati tabele oz. nize podatkov. V tej funkciji sodeluje indeksni register pri določanju naslovov podatkov, do katerih želi CPE dostopati. Ta operacija se imenuje »indeksno naslavljanje«. Obstaja več različnih oblik indeksnega naslavljanja. Opisali bomo eno najpogostejših oblik in pokazali njeno vrednost v določenih programskih primerih.

Osnovna zamisel indeksnega naslavljanja je, da je dejanski oz. efektivni naslov operanda, ki ga določa neka instrukcija, enak vsoti tistega dela instrukcije, ki določa naslov operanda in vsebine indeksnega registra. Za ilustracijo si poglejmo primer. Indeksni register je 8-bitni in vsebuje število 4_{16}. Predpostavimo tudi, da CPE izvaja instrukcijo polnjenja akumulatorja z uporabo indeksnega naslavljanja (simbol LDA,X). To instrukcijo bi lahko shranili v spomin na naslednji način:

Tabela 1.23: Primer indeksnega naslavljanja

Naslov v spominu	Spominska beseda	Komentar
0200	6F	Koda za LDA,X
0201	50	Nižji bajt naslova 0450
0202	04	Višji bajt naslova 0450

Za izvedbo te instrukcije CPE prebere kodo 6F na lokaciji 0200, ki določa, da se izvaja operacija LDA, X, pri kateri je treba podatkovno besedo naložiti v akumulator. Dejanski naslov te podatkovne besede se določi tako, da se iz spominskih lokacij 0201 in 0202,

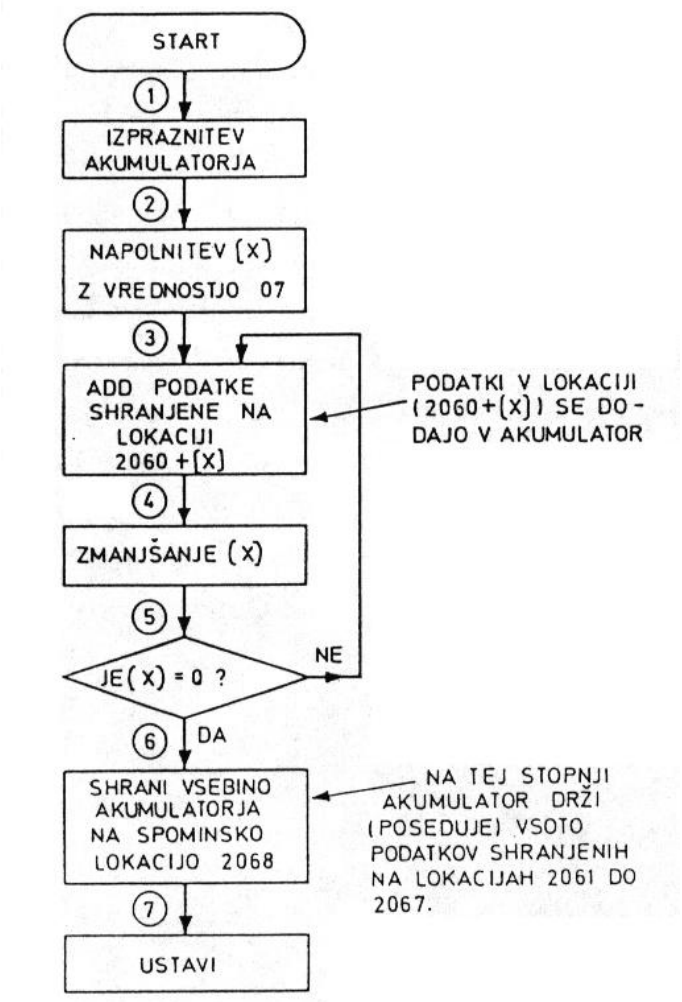
imenovanih tudi bazni naslovi, vzame del instrukcije, ki določa naslov operanda in se jih doda vsebina indeksnega registra X. Tako je

$$\begin{aligned} \text{dejanski naslov} &= \text{bazni naslov} + [X] \\ &= 0450 + [X] \\ &= 0454 \end{aligned}$$

S tega naslova (0454) CPE prebere podatkovno besedo, ki jo vpiše v akumulator.

Če se vsebina X spremeni in to instrukcijo ponovimo, se prebere iz spomina druga podatkovna beseda. Zaradi tega je indeksno naslavljanje zelo koristno pri obdelavi tabel podatkov, ki so shranjeni v spominu. To bomo pokazali na naslednjem primeru enostavnega programa.

Primer:
Tabelo sedmih podatkovnih besed želimo shraniti na spominske lokacije od 2061 do 2067. Narišite diagram poteka za program, ki bo seštel teh sedem podatkovnih besed in shranil seštevek v spomin na naslov 2068. Uporabite indeksno naslavljanje.

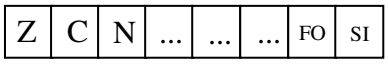


Slika 1.53: Uporaba indeksnega naslavljanja

Rešitev:
Zahtevani diagram poteka je prikazan na sliki 1.53. Pred začetkom programa izpraznimo akumulator. Indeksni register napolnimo z vrednostjo 7. Instrukcije v blokih 3, 4 in 5 se izvajajo skupaj sedemkrat, preden se izvajanje programa premakne na blok 6. Prištevanje se opravi vsakokrat s podatkom na drugi lokaciji. Prične se s podatkom na lokaciji 2067, zaključi pa s podatkom na lokaciji 2061. Na ta način je bila kompletna tabela podatkov prišteta k akumulatorju.

STATUSNI REGISTER
Statusni register, imenovan tudi »procesni statusni register« ali »register stanj« je sestavljen iz posameznih bitov. Proizvajalec določi vsakemu svoj pomen. Ti biti se imenujejo zastavice (kazalci) ali »angl. flag« in služijo za označevanje specifičnega mikroprocesorskega stanja. Vrednost nekaterih zastavic lahko spremljamo programsko in njihova vrednost lahko določi naslednjo sekvenco, ki naj se izvaja. Dve izmed najpomembnejših zastavic sta Z, »Zero Flag« in »Carry Flag« ali C. Vrednost Z bo vedno označevala ali je instrukcija, ki se je izvedla, ustvarila rezultat nič. Običajno krmilni signal mikroprocesorja postavi Z v visoko stanje kadar je rezultat neke operacije nič in postavi Z v nizko stanje, ko rezultat neke instrukcije ni enak nič.
Vrednost C vedno označuje, ali je predhodna instrukcija ustvarila rezultat, ki je presegel velikost mikroprocesorske besede. Npr. kadarkoli seštevek dveh osembitnih podatkovnih besed ustvari vsoto, ki presega 8 bitov, bo C postavljen na 1. Če pa seštevek ne presega 8-bitnega zapisa, bo C = 0. C lahko smatramo kot deveti bit aritmetičnega rezultata.

Nekateri mikroprocesorji imajo enega ali več bitov statusnega registra povezanih z zunanjimi priključki. To je ilustrirano na sliki 1.54 za bita 6 in 7. Bit 6 je zastavični izhod (FO – Flag Output), ki ga mikroprocesor lahko uporablja za krmiljenje zunanje naprave. Logično stanje na tem izhodu se lahko tekom programa spreminja. Visoko stanje na tem izhodu lahko služi za vklop zunanje naprave.



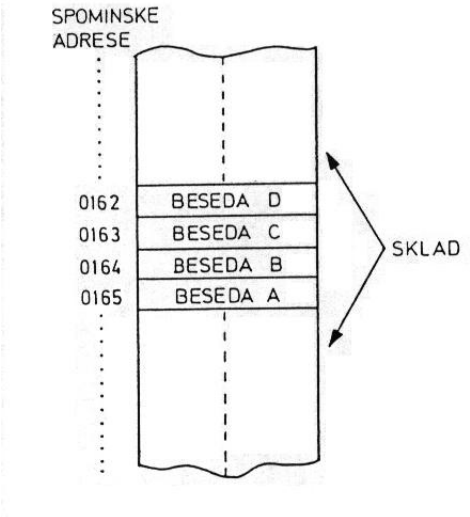
Slika 1.54: Primer razporeditve bitov v statusnem registru

Bit 7 SI (Sense Input) je vhod mikroprocesorja, ki omogoča detektiranje spremembe logičnega stanja in ustrezno spreminjanje teka programa. Tako je omogočena neposredna komunikacija s CPE.

Večina mikroprocesorjev omogoča izvajanje instrukcij, ki se izvajajo v odvisnosti od stanja bitov statusnega registra (npr. JPZ).
Drugi primer je instrukcija JCS (Jump on Carry Set) ali JCC (Jump on Carry Cleared).
Ostale bite statusnega registra bomo obravnavali pri uporabi mikroprocesorja.

REGISTER KAZALCEV SKLADA (STACK POINTER REGISTER)
Preden definiramo funkcijo tega registra, moramo definirati sklad podatkov. To je del RAM-a, ki je rezerviran za začasno shrambo in iskanje podatkov, predvsem vsebine mikroprocesorjevih internih registrov. To področje spomina imenujemo »stack«, ki deluje na naslednji način:

1. Vsakokrat, ko se mora neka beseda shraniti v to območje RAM-a, se postavi na spominsko lokacijo, ki je za eno manjša od prejšnje besede, shranjene v skladu podatkov. Da bi to lahko ilustrirali, si poglejmo sliko 1.55.
2. Besede, shranjene v bloku podatkov, se odčitavajo z bloka v obratnem vrstem redu kakor so bile vanj vpisane. Na sliki 1.55 to pomeni, da se mora beseda D brati prva, nato besedi B in C ter končno beseda A.
3. Ko se beseda prebere iz bloka, je njena lokacija na razpolago za novo vpisovanje.

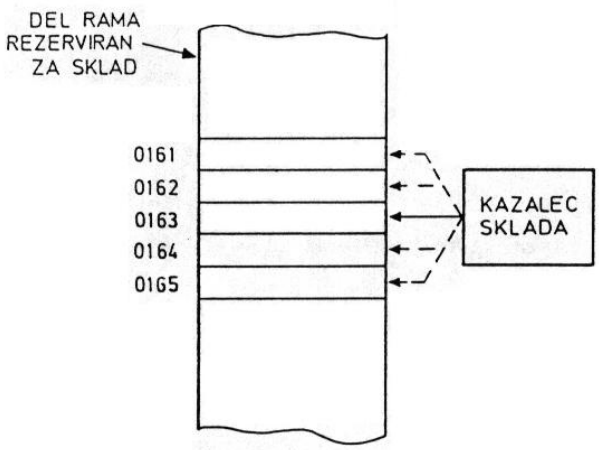


Slika 1.55: Organizacija sklada

Takšen način vpisovanja podatkov v sklad in branja podatkov iz njega imenujemo tudi LIFO – (zadnji noter prvi ven – LIFO stack).

Prostor v RAM-u in njegov obseg, ki je rezerviran za sklad, se med posameznimi tipi mikroprocesorjev razlikuje.

Sedaj lahko pojasnimo vlogo registra kazalcev sklada (SP). Ta register deluje kot posebni spominski naslovni register, ki se uporablja le za del RAM-ovega sklada. Kadarkoli je treba besedo shraniti v sklad, shranimo na naslov, ki je zapisan v SP. Enako velja za branje iz sklada. Podatek beremo na tisti lokaciji, ki jo določa SP. Vsebino SP registra postavi v začetno stanje programer na začetku programa. Potem se SP avtomatsko zmanjša, ko se beseda shrani v skladu in poveča, ko se beseda prebere iz sklada. To izvaja krmilna enota mikroprocesorja avtomatsko.



Slika 1.56: Register kazalcev sklada kaže na naslednjo prosto lokacijo v skladu

Operacija SP je prikazana na sliki 1.56. Predpostavimo, da je zadnja operacija sklada hranila neko besedo na lokaciji 0164. Po tej operaciji se je SP zmanjšal na 0163, kar označuje, da je lokacija 0163 naslednja razpoložljiva lokacija v skladu. To je simbolično prikazano na sliki, kjer SP kaže na lokacijo 0163. Če je naslednja operacija SHRANI, bo beseda shranjena na lokaciji 0163, SP pa se bo zmanjšal na 0162. Druga operacija shranjevanja bo postavila besedo na 0162 in zmanjšala vrednost SP na 0161. Sedaj pa predpostavimo, da se izvaja operacija branja iz sklada. Kazalec sklada kaže na lokacijo 0161 in se avtomatično poveča na 0162 in CPE odčita tam shranjeno besedo. Podobno se ob naslednjih operacijah branja SP poveča na 0163, 0164 itd.

Za povečevanje in za zmanjševanje SP skrbi mikroprocesor. Programer pa mora poskrbeti za začetno stanje SP. Nikdar ne sme biti postavljen na lokacije izven območja RAM-a, ki je rezervirano za sklad. Drugače lahko pride do prepisovanja ene besede čez drugo (prepiše se podatek ali pa ukaz). Poleg tega se mora programer prepričati, da je jemal iz sklada v obratnem vrstnem redu kot je vanj vlagal.

2 Spominske enote

V tem poglavju bomo obravnavali različne vrste spominskih enot, ki jih dobimo na trgu v standardni obliki integriranih vezij.
Vsako sekvenčno vezje ima že ima tudi funkcijo spomina saj vsak flip-flop ali zapah shranjuje en bit informacije. Kljub temu pa običajno z imenom spomin razumemo bite, ki so strukturirano shranjeni v dvodimenzionalnem polju, kjer lahko v določenem trenutku dostopamo do izbranih bitov ene vrstice.

Spominske enote lahko uporabimo za reševanje različnih nalog. Tako lahko npr. uporabimo bralni spomin v centralni procesni enoti za določitev osnovnih korakov, ki so potrebni za izvedbo posameznega ukaza. Hitri »statični spomin« lahko uporabljamo v funkciji »cache« spomina za shranjevanje ukazov in podatkov, ki jih CPE v določenem trenutku uporablja. Glavni spomin (dinamični spomin) mikroračunalnika je običajno obsežnejši (nekaj 100Mbitov) in vanj lahko shranimo celoten operacijski sistem, program in podatke.

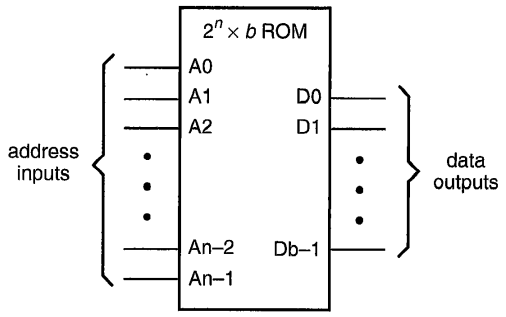
Uporaba spominskih enot pa ni omejena le na mikroprocesorsko tehniko oz. samo na digitalne sisteme. V javnem telefonskem sistemu se uporablja ROM spomin za izvajanje transformacij digitaliziranih glasovnih signalov. Veliko prenosnih CD predvajalnikov uporablja dinamični spomin za branje in shranjevanje nekaj sekund zapisa vnaprej, pri čemer moramo doseči hitrost prenosa digitalnega zapisa glasbe vsaj 1,4 Mbit/s.
Poleg tega imamo vrsto audio/video naprav, ki uporabljajo spomin za začasno shranjevanje digitaliziranih signalov za nadaljnjo obdelavo in izboljšavo s postopkom digitalnega signalnega procesiranja.

Najprej si bomo ogledali zgradbo in primere uporabe ROM spomina. Nadaljevali bomo z opisom dveh tipov bralno-pisalnega spomina. Na kratko si bomo ogledali tudi notranjo zgradbo spominskih enot.

V drugem delu bomo obravnavali CPLD in FPGA vezja. Te enote so sorodne spominskim enotam v tem, da vsebujejo veliko osnovnih logičnih sklopov, ki jih lahko uporabljamo v najrazličnejše namene. Ker vanje lahko zelo hitro programiramo različne logične funkcije, so postali zelo priljubljeni elementi v sodobni digitalni tehniki.

2.1 ROM spomin

Bralni spomin (ROM) je v splošnem kombinacijsko vezje, ki ima *n* vhodov in *b* izhodov (Slika 2.1). Vhodi so naslovi, ki jih označujemo z A0, A1,...,An-1. Na izhodih, ki jih najpogosteje označujemo z D0, D1,...Db-1, dobimo podatke. V ROM-u je shranjena kombinacijska tabela z *n* vhodi in *b* izhodi. Tabela 1.1 ima npr. tri vhode in štiri izhode. Vanjo lahko zapišemo osem štiribitnih besed. Če zanemarimo zakasnitvene čase, potem so stanja na izhodih v vsakem trenutku enaka stanjem bitov neke vrstice, ki jo izberemo z logičnimi stanji na vhodnih oz. na naslovnih linijah.



Slika 2.1 ROM z obsegom 2^n x *b*

Ker je ROM pravzaprav kombinacijsko vezje, ga le pogojno uvrščamo med spominska vezja. Glede na vrsto logičnih operacij, ki jih z njim lahko izvajamo, se v ničemer ne razlikuje od kombinacijskih vezij. Lahko rečemo, da mu funkcijo spomina določi proizvajalec, ki vanj »zapiše« podatke v postopku izdelave. Poleg tega se razlikuje od mnogih ostalih spominskih vezij po sposobnosti ohranjanja zapisane vsebine tudi po izklopu napajalne napetosti.

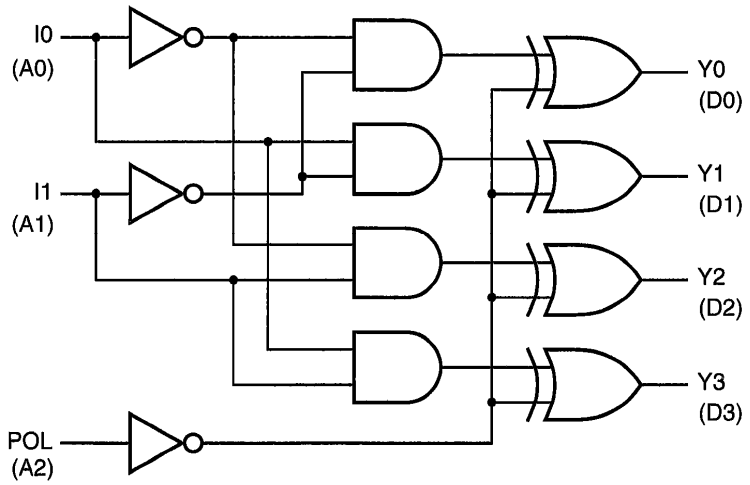
2.1.1 Uporaba ROM-a za izvedbo poljubnih kombinacijskih logičnih funkcij.

V izjavnostni tabeli 2.1 so zapisana stanja splošnega 2-4 dekodirnega vezja. Tretji vhod (A2) omogoča izračun prvega komplementa zapisane besede.

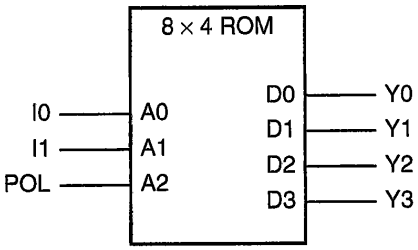
Tabela 2.1

Inputs			*Outputs*			
A2	***A1***	***A0***	***D3***	***D2***	***D1***	***D0***
0	0	0	1	1	1	0
0	0	1	1	1	0	1
0	1	0	1	0	1	1
0	1	1	0	1	1	1
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Na sliki 2.2 je prikazano diskretno vezje za rešitev omenjene naloge. Če torej povzamemo, lahko nalogo rešimo z uporabo 8 x 4 ROM-a (Slika 2.3) ali pa z uporabo diskretnih logičnih elementov. Pri uporabi ROM-a imamo več možnosti pri izbiri vrstnega reda vrstic in stolpcev za izvedbo določene naloge. To dosežemo z ustreznim označevanjem oz. z dodelitvijo ustreznih funkcij vhodov in izhodov. Če npr. zamenjamo bite v stolpcih D0 in D3, potem dobimo izjavnostno tabelo za drug ROM (Tabela 2.2). Kljub temu pa lahko še vedno uporabimo ta ROM za rešitev prejšnje naloge, če medsebojno zamenjamo tudi oznaki izhodov Y0 in Y3. Pri zamenjavi posameznih vrstic je treba temu ustrezno spremeniti oznake vhodov.



Slika 2.2 Primer dekodirnega vezja



Slika 2.3 Izvedba dekodirnega vezja s pomočjo 8 x 4 ROM-a

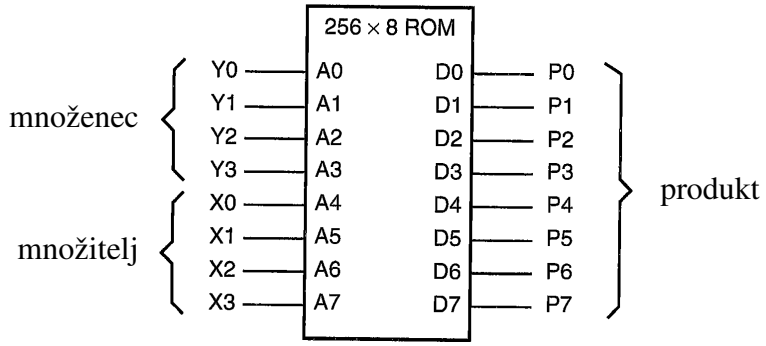
Tabela 2.2 Izjavnostna tabela s spremenjenim vrstnim redom podatkovnih vrstic

Inputs			*Outputs*			
A2	***A1***	***A0***	***D3***	***D2***	***D1***	***D0***
0	0	0	1	1	1	0
0	0	1	0	0	0	1
0	1	0	1	1	0	1
0	1	1	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	1	0	0
1	1	0	0	1	1	1
1	1	1	1	0	0	0

Naslednji primer uporabe ROM-a je izvedba množenja dveh štiribitnih binarnih števil brez upoštevanja predznaka. Prednost uporabe ROM-a pred standardnimi logičnimi vezji je možnost, da napišemo program v zbirniku ali v nekem višjem programskem jeziku, ki nam izračuna vrednosti tabele 2.3 in jih potem v shranimo v ROM na sliki 2.4.

Tabela 2.3 Izvedba 4 x 4 bitnega množilnika z ROM-om

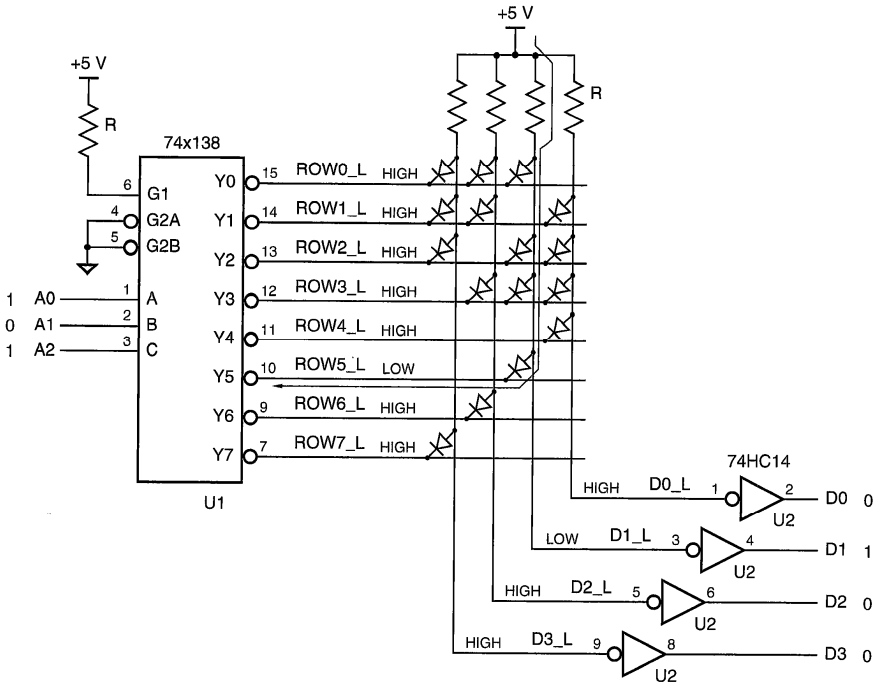
```
00: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
10: 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
20: 00 02 04 06 08 0A 0C 0E 10 12 14 16 18 1A 1C 1E
30: 00 03 06 09 0C 0F 12 15 18 1B 1E 21 24 27 2A 2D
40: 00 04 08 0C 10 14 18 1C 20 24 28 2C 30 34 38 3C
50: 00 05 0A 0F 14 19 1E 23 28 2D 32 37 3C 41 46 4B
60: 00 06 0C 12 18 1E 24 2A 30 36 3C 42 48 4E 54 5A
70: 00 07 0E 15 1C 23 2A 31 38 3F 46 4D 54 5B 62 69
80: 00 08 10 18 20 28 30 38 40 48 50 58 60 68 70 78
90: 00 09 12 1B 24 2D 36 3F 48 51 5A 63 6C 75 7E 87
A0: 00 0A 14 1E 28 32 3C 46 50 5A 64 6E 78 82 8C 96
B0: 00 0B 16 21 2C 37 42 4D 58 63 6E 79 84 8F 9A A5
C0: 00 0C 18 24 30 3C 48 54 60 6C 78 84 90 9C A8 B4
D0: 00 0D 1A 27 34 41 4E 5B 68 75 82 8F 9C A9 B6 C3
E0: 00 0E 1C 2A 38 46 54 62 70 7E 8C 9A A8 B6 C4 D2
F0: 00 0F 1E 2D 3C 4B 5A 69 78 87 96 A5 B4 C3 D2 E1
```



Slika 2.4 Izvedba štiribitnega množilnika z 256 x 8 ROM-om

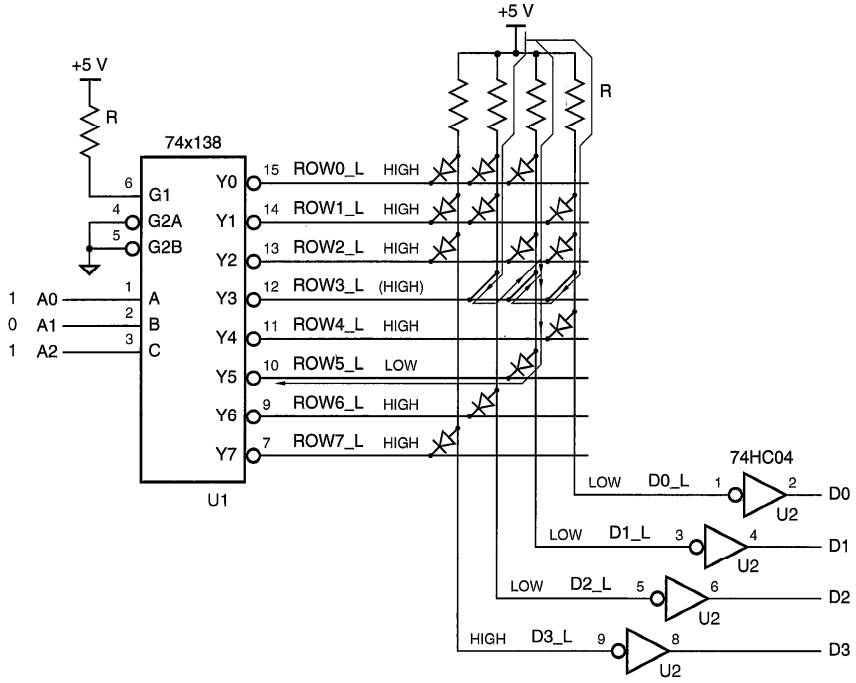
2.1.2 Zgradba ROM-a

Obstaja več načinov zapisovanja podatkov ROM. Najpogosteje je logična ena določena z vezavo diode ali tranzistorja na izbranem mestu v matriki. Če diode ali tranzistorja ni na nekem mestu to pomeni, da je tam vpisana logična nič. Na sliki 2.5 je prikazana ena od možnih izvedb 8 x 4 ROM-a z uporabo dekodirnega vezja in diod. Vsaka vhodna kombinacija dekodirnega vezja izbere torej eno izhodno vrstico, ki tako določa shranjeno besedo v ROM-u. Na sliki je prikazano stanje, ko je A2, A1, A0 = 101 in je izbrana vrstica ROW_5. Vertikalne povezave imenujemo bitne linije in posredujejo podatke na izhode D3 – D0. Pri izbiri določene vrstice in ob prisotnosti diode med bitno linijo in vrstice pade potencial bitne linije na logično nič. Ker je v vrstici ROW5_L samo ena dioda, se bitna linija D1_L postavi v stanje logične nič, ostale linije pa ostanejo na logični ena. Zaradi inverterjev na izhodu dobimo v tem primeru na izhodu besedo 0010.
Vsako presečišče bitne in vrstične oz. besedne linije določa en bit »spomina«. Če je na presečišču prisotna dioda, dobimo logično ena, v nasprotnem primeru je na tem mestu logična nič.



Slika 2.5 Izvedba 8 x 4 ROM-a z diodami in dekodirnim vezjem 74x138

Postavitev diod na sliki 2.5 rešuje problem, ki smo si ga zastavili na začetku poglavja (Tabela 2.1). Z uporabniškega stališča je takšna izvedba ROM-a precej nerodna, saj rabimo 3-8 dekodirno vezje in veliko diod. V nadaljevanju si bomo pogledali bolj uporabno strukturo ROM-a.

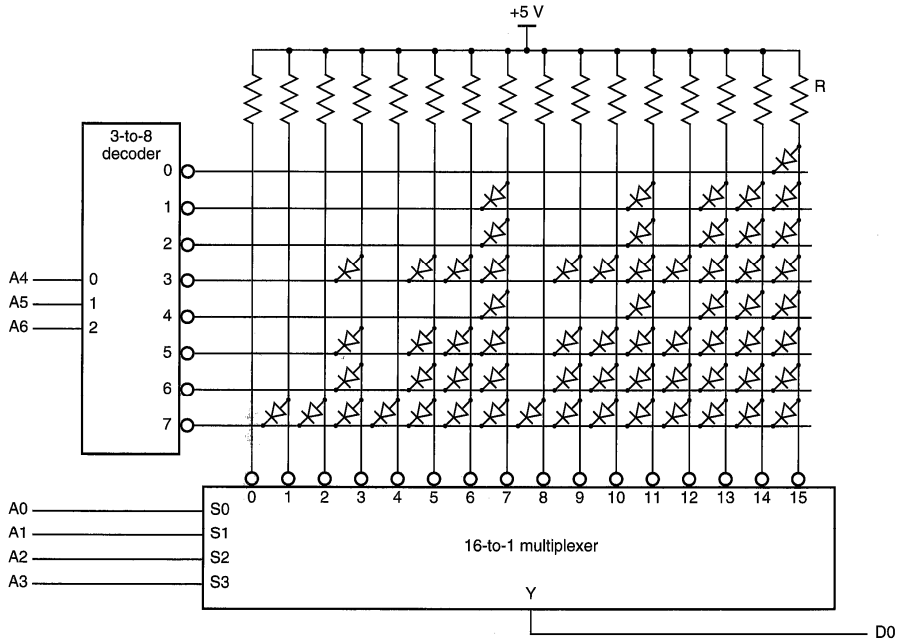


Slika 2.6 Napačno delovanje ROM-a v primeru kratkostičnih povezav

2.1.3 Dvodimenzionalno dekodiranje

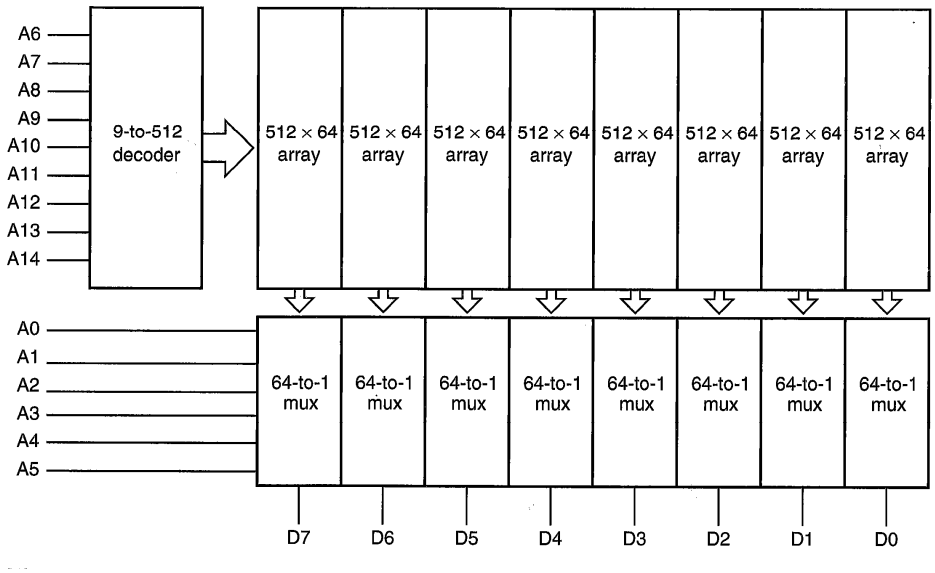
Če želimo sestaviti ROM s kapaciteto 128 x 1 z uporabo vezij, ki so bila predstavljena v prejšnjem poglavju, potem potrebujemo veliko število logičnih vrat. V takem primeru izberemo dvodimenzionalno dekodiranje pri čemer zmanjšamo velikost dekodirnega vezja na vrednost kvadratnega korena števila naslovnih linij.

Pri dvodimenzionalnem dekodiranju sestavimo spominske celice ROM-a v obliki polja, ki naj bo čim bolj podobno kvadratu. Na sliki 2.7 je prikazana ena od možnih izvedb ROM-a z velikostjo 128 x 1. Zgornje tri naslovne bite A6-A4 uporabimo za izbiro posamezne vrstice v kateri je shranjenih 16 bitov. Celotni naslov vrstice je določen z nizom (A6, A5, A4,0,0,0). Ko je naslov na vodilu veljaven, preberemo vseh 16 bitov v izbrani vrstici. Multipleksor s 16. vhodi nato izbere želeni bit v izbrani vrstici in ga posreduje na izhod D0.



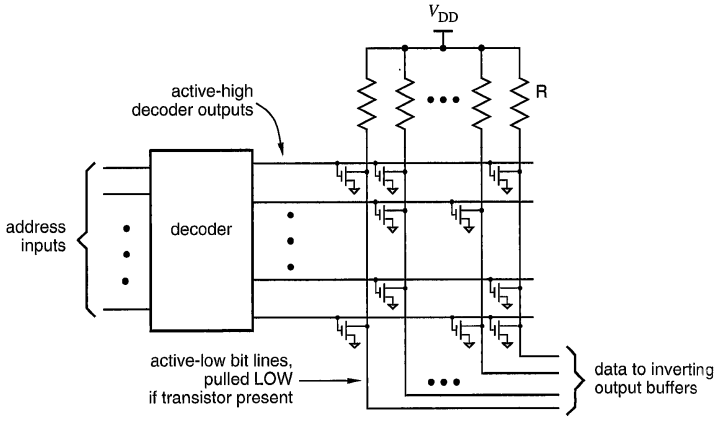
Slika 2.7 Struktura 128 x 1 ROM-a z uporabo dvodimenzionalnega dekodiranja

Postavitev diod na sliki 1.7 ni naključna. S to postavitvijo lahko izvedemo kombinacijsko logiko, za kar bi potrebovali 35 IN vrat s po štirimi vhodi.
1M x 1 ROM bi lahko zgradili z 10 na 1024 dekodirnim vezjem in s 1024 vhodnim multipleksorjem, kar ni ravno enostavno, vendar veliko bolj, kot če bi uporabili enodimenzionalnega dekodiranja. Po drugi strani je dvodimenzionalno dekodirno vezje tudi lažje izdelati, ker ima obliko kvadrata, enodimenzionalna dekodirna vezja pa so ozka in zelo dolga.
Pri večbitnih izhodnih besedah je treba sestaviti več sklopov v eno celoto. Na sliki 2.8 je prikazana zgradba ROM-a s kapaciteto 32K x 8.



Slika 2.8 Principialna zgradba ROM-a s kapaciteto 32K x 8

V ROM-ih, ki so izdelani v MOS tehnologiji (Slika 2.9), diode nadomeščajo tranzistorji. Izhodne linije dekodirnega vezje so aktivne, ko so v logičnem stanju ena. Vsi tranzistorji v izbrani vrstici so takrat vklopljeni in ustrezna bitna (izhodna) linija je v stanju logične nič. Na podoben način delujejo ROM-i, ki imajo vgrajene bipolarne tranzistorje.



Slika 2.9 MOS tranzistor v vlogi spominskega elementa v ROM-u

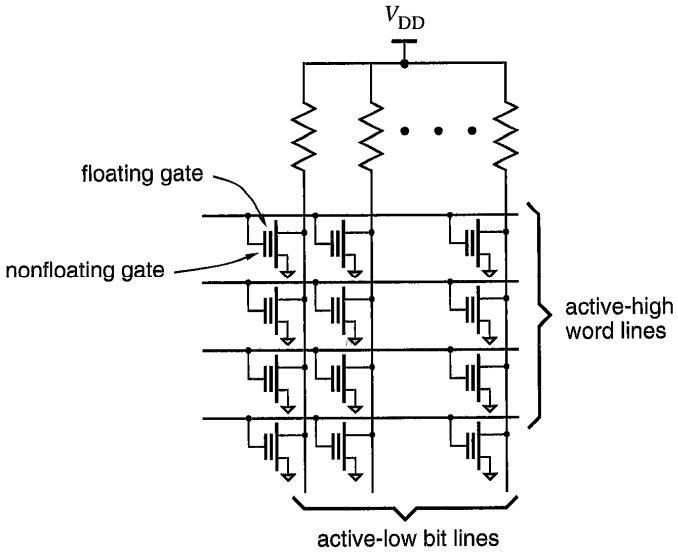
2.1.4 Praktične izvedbe ROM-ov

Proizvajalci ROM-ov uporabljajo različne postopke za vpisovanje podatkov. Pri prvih izvedbah se je najpogosteje uporabljala tehnika programiranja z masko, s katero se določi mesta kjer so vertikalne in horizontalne linije povezane in tiste brez povezav. Uporabnik je torej podal proizvajalcu vsebino, ko jo je želel imeti v ROM-u. Postopek za izdelavo maske je sorazmerno drag, zato se danes uporablja le še pri zelo velikih ROM-ih.

Pri manjših kapacitetah spomina uporabljamo PROM, ki je po strukturi podoben ROM-u z masko. PROM je izdelan tako, da ima pred uporabo vse diode ali tranzistorje na povezovalnih

mestih v takšnem stanju, da prevajajo. To pomeni, da je na teh mestih vpisana logična ena. S pomočjo programirne naprave postavimo želene bite v stanje logične nič. Pri bipolarnih izvedbah se postopek izvede tako, da se z naslovnimi in podatkovnimi linijami določi želeni bit nato pa s priključitvijo visoke napetosti (10-30 V) na posebnem vhodu prekinemo povezavo, ki je izvedena s tanko žico, podobno kot pri klasični varovalki. Prvi PROM-i, ki so bili izvedeni z bipolarno tehniko, so bili še zelo nezanesljivi. Pri programiranju ni prišlo vedno do prekinitve povezav, ali pa se je povezava vzpostavila preko uparjene plasti, ki je nastala ob prekinitvi (uparjanju) povezovalne žice. Danes so te težave že odpravljene in ta tehnologija se uporablja tudi pri izdelavi PLD vezij (Programable Logic Devices).

Programirljivi in zbrisljivi ROM (EPROM) programiramo na enak način kot PROM-e, brišemo pa jih tako, da jih izpostavimo UV svetlobi. Pri tem se vsi biti postavijo v stanje logične ena. Pri tem postopku ne gre za ponovno vzpostavitev fizičnih vezi, ampak uporabimo tehnologijo »plavajočih vrat MOS tranzistorja« (Slika 1.10). Vsak tranzistor na posameznem spoju ima dvoje vrat. Plavajoča vrata so obdana z zelo kakovostnim izolacijskim materialom. Druga vrata pa so pritrjena na horizontalne izbirne linije. Pri programiranju EPROM-a priključimo visoko napetost na »fiksna« vrata na mestu, kjer želimo vpisati logično nič. To povzroči kratkotrajen preboj izolacijskega materiala in na plavajaočih vratih se nakopiči negativni naboj. Ko napetost odstranimo, se naboj na plavajočih vratih ohrani. Med poznejšimi bralnimi operacijami negativni naboj preprečuje vklop MOS tranzistorja na izbrani liniji. Proizvajalci EPROM-ov zagotavljajo, da pravilno programiran bit ohrani po desetih letih še vedno 70% prvotnega naboja, čeprav vezje obratuje pri temperaturi 125°C.



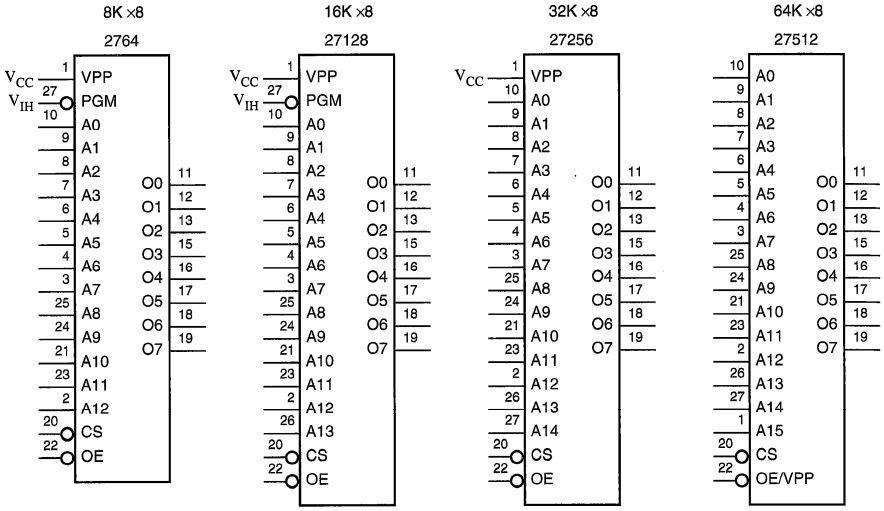
Slika 2.10 MOS tranzistorji s »plavajočo« maso v EPROM-u

Če izolacijski material, ki obdaja »plavajoča vrata« tranzistorja, izpostavimo UV svetlobi z določeno valovno dolžino za 5-20 minut, potem postane delno prevoden.

EPROM-e najpogosteje uporabljamo za shranjevanje programov med razvojem ciljnega vezja. V večini današnjih PROM integriranih vezjih so dejansko vgrajeni EPROM-i. Ker tovrstna vezja nimajo odprtine za brisanje vsebine, jih imenujemo tudi ROM vezja za enkratno programiranje. (OTP – One Time Programmable ROM).

EEPROM je podoben EPROM-u, le brisanje pri njem poteka s pomočjo električne napetosti. Izolacijske plasti okrog plavajočih vrat so tanjše in bit brišemo tako, da na fiksna vrata pritisnemo visoko napetost nasprotne polaritete kot pri programiranju. Pri EEPROM-ih z večjo kapaciteto (1 Mbit ali več) lahko brišemo le posamezne spominske bloke (16 - 64 Kbytov). Brisanje se odvija »bliskovno«.

Pisanje EEPROM-a se odvija počasneje kot pa branje. Zaradi tega ga ne uporabljamo namesto bralno-pisalnega spomina. Ker je izolacijska plast vrat zelo tanka, lahko pride pri večkratnem vpisovanju do poškodbe. Skratka, EEPROM-i so predvideni za končno število vpisov (vsaj 10000). Najpogosteje jih uporabljamo za shranjevanje podatkov, ki jih ne želimo izgubiti ob izklopu napajalne napetosti. Standardni EPROM-i za razširitev mikroprocesorskih enot so prikazani na sliki 2.11.



Slika 2.11 Logični simboli za standardne izvedbe EPROM-ov v 28-pinskem ohišju

2.2 Bralno-pisalni spomin

Pri tej vrsti spomina lahko podatke kadarkoli vpišemo in beremo. Potreben čas za branje oz. pisanje podatka je neodvisen od lokacije podatka v RAM-u. Po vpisu podatka v statični RAM (SRAM) se le ta v njem lahko ohrani do izklopa napajalne napetosti. Pri tem seveda predpostavimo, da v tem času ni prišlo do ponovnega vpisa z drugo vsebino.
Pri dinamičnem RAM-u (DRAM) moramo shranjene podatke periodično osveževati. To izvedemo tako, da vpisan podatek najprej preberemo, nato pa ga ponovno vpišemo. Če vsebine spominske celice ne »osvežimo« se njena vsebina kljub prisotnemu napajanju čez nekaj časa izgubi. V tem poglavju si bomo podrobneje ogledali obe vrsti spomina.
Pri večini RAM-ov se njihova vsebina ob izklopu napajalne napetosti izgubi. Pri nekaterih lahko njihovo vsebino po izklopu ohranimo tako, da uporabimo litijev elektrokemijski člen, ki vzdržuje napajanje tudi do deset let. Sposobnost pomnjenja brez prisotnosti napajalne napetosti so imeli tudi prvi RAM-i, ki so bili izdelani s feritnimi jedri.
V zadnjem času so razvili ti. feroelektrične RAM-e, ki v integriranem vezju združujejo elektronske in magnetne elemente. S tem dosežejo, da se njihova vsebina ohranja tudi ob izklopu električne napetosti.

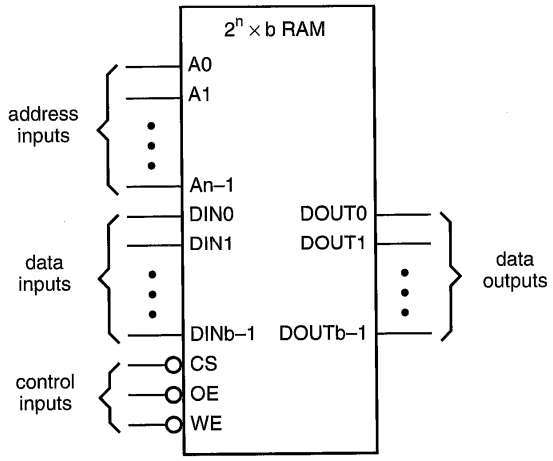
2.2.1 Statični RAM (SRAM)

RAM ima poleg naslovnih linij, izhodnih podatkovnih linij in krmilnih linij tudi vhodne podatkovne linije. Na sliki 2.12 je prikazan preprost statični RAM s kapaciteto 2^n x *b* bitov. Krmilni vhod so podobni kot pri ROM-u. Dodan je le krmilni vhod »Write-Enable« (WE). Ko je na omenjenem vhodu aktivno stanje, se podatki vpisujejo na izbrane lokacije.

Spominske lokacije v statičnem RAM-u so izvedene z D zapahi. Ko je krmilni vhod WE aktiven, so zapahi za izbrano spominsko lokacijo sproščeni in podatki se prenesejo naprej. V spominsko lokacijo se zapiše tisti podatek, ki je prisoten v trenutku, ko se zapah zapre.

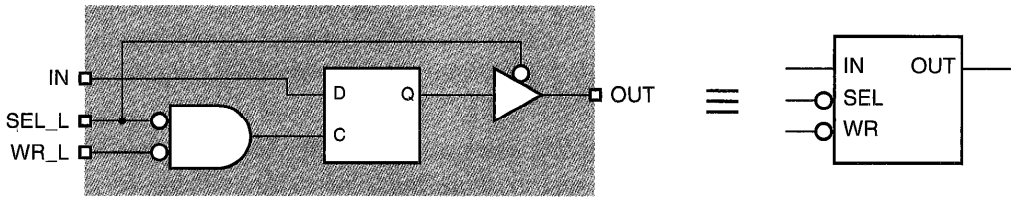
Bralna operacija poteka pri aktivnem stanju na vhodih CS in OE. Izbrani podatki, ki so določeni s stanjem na vhodih za naslov, se prenesejo preko zapaha na izhode DOUT.

Pri pisalni operaciji se izbrana beseda, ki je prisotna na podatkovnih vhodih DIN, ob aktiviranju krmilnih vhodov CS in WE oz. ob sprostitvi zapahov prenese na izbran naslov, kjer se shrani.



Slika 2.12 Preprost statični RAM s kapaciteto 2^n x *b* bitov

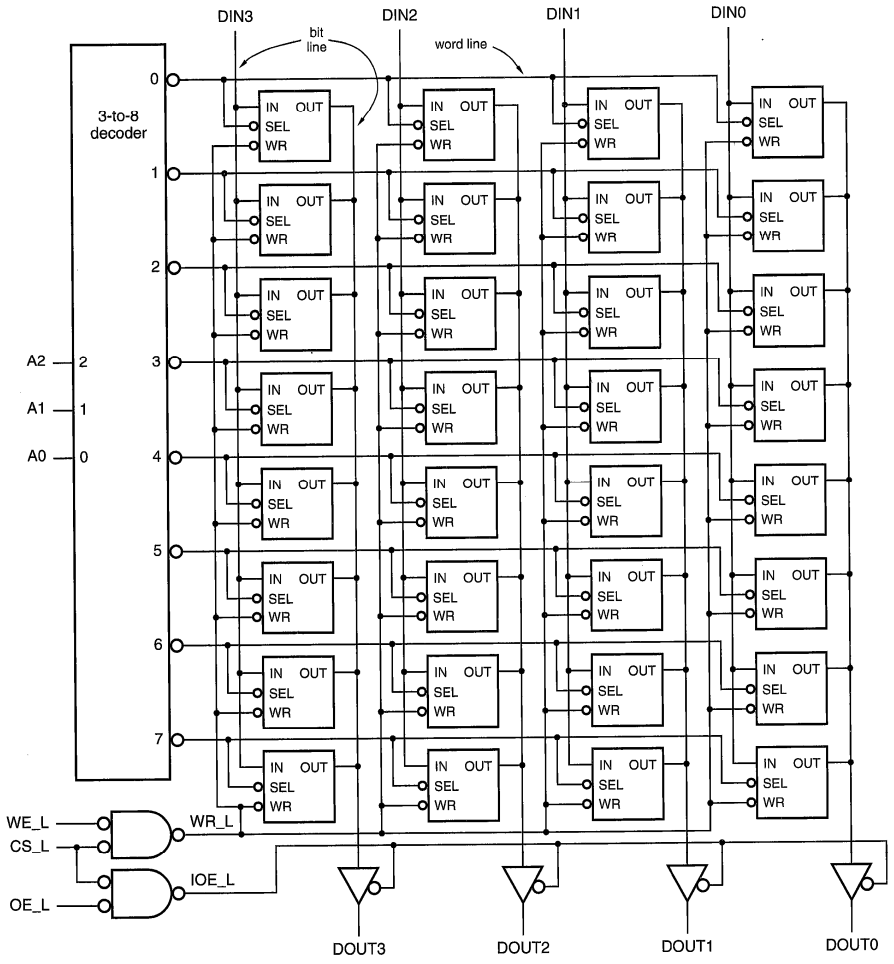
Principialno zgradbo posamezne spominske lokacije statičnega RAM-a vidimo na sliki 2.13. Osnovna spominska celica je D-zapah. Ko je na vhodih SEL_L in WR_L logična nič, se zapah sprosti in vanj lahko zapišemo podatek, ki je na vhodu IN.



Slika 2.13 Nadomestno vezje celice statičnega RAM-a

Pri združevanju večjega števila celic (npr. 8 x 4 SRAM) rabimo dodatno krmilno logiko, ki je prikazana na sliki 2.14. V ta namen je uporabljeno preprosto ROM dekodirno vezje. Pri določenem naslovu na vhodu dekodirnega vezja se sprosti izbrana vrstice spominskega polja. Vezje na sliki 2.14 je nekoliko poenostavljeno, vendar iz njega lahko razberemo osnovne značilnosti statičnega RAM-a:

1. Pri bralni operaciji je izbira podatka, podobno kot pri ROM-u, odvisna le od kombinacijske logike pri naslavljanju, podobno kot v ROM-u. Če ob sproščenem izhodnem podatkovnem vodilu zamenjamo naslov besede, ne povzročimo nedovoljenega stanja. Čas dostopa je določen s časovnim intervalom, ko se na naslovnih vhodih ustalijo nova logična stanja.
2. Med pisalnimi operacijami se vhodni podatki shranjujejo v »zapahe«. Pri tem je treba paziti, da so podatki na vhodih postavljeni in veljavni prej preden aktiviramo krmilni vhod WR_L.
3. Med pisalno operacijo morajo biti stanja na naslovnih vhodih ustaljena preden aktiviramo vhod WR_L. V nasprotnem primeru lahko pride do vpisovanja podatkov na neprava mesta zaradi prenihanj na SEL_L linijah ob spremembi naslova na vhodu dekodirnega vezja.
4. Pomembno je, da se aktiviranje WE_L izvrši istočasno z aktiviranjem CS_L in WE_L. Pisalni cikel se torej začne ob aktiviranju vhodov CS_L in WE_L in se konča takoj, ko je katerikoli od obeh negiran.



Slika 2.14 Zgradba statičnega RAM-a s kapaciteto 8 x 4 bitov.

2.2.1.1 Časovni intervali pri pisanju oz branju statičnega RAM-a

Na sliki 2.15 je prikazan časovni diagram poteka signalov na vhodih in izhodih pri bralni operaciji statičnega RAM-a. Proizvajalci podajajo različne časovne intervale pri branju in si jih bomo v nadaljevanju podrobneje ogledali:

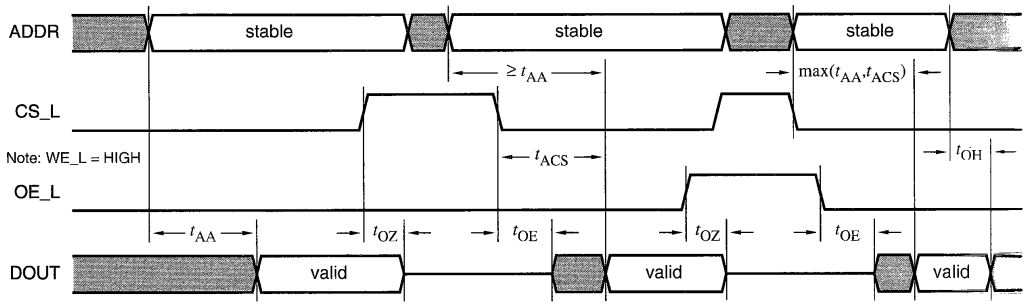
- t_{AA} – Predpostavimo, da sta krmilna vhoda OE in CS že aktivna. Dostopni čas t_{AA} je čas, ki preteče od trenutka, ko je na naslovnem vodilu veljaven naslov do trenutka, ko se postavijo podatki na podatkovnem vodilu. Ko proizvajalci govorijo o »70 ns SRAM-u«, potem najpogosteje uporabljajo vrednost t_{AA}.

- t_{ACS} – Dostopni čas glede na spremembo signala CS. Predpostavimo, da so naslovi na vodilu veljavni in da je OE vhod že aktiven. t_{ACS} je čas, ki je potreben, da se po aktiviranju signala CS vzpostavijo podatki na vodilu.

- t_{OE} – je čas, ki je potreben, da izhodi RAM-a po aktiviranju krmilnih vhodov OE in CS preidejo iz visokoimpedančnega stanja.

- t_{OZ} – označuje časovni interval, ki je potreben, da se po spremembi stanj na OE in CS krmilnih vhodih, izhodni podatkovni ojačevalniki postavijo v stanje visoke impedance.

- t_{OH} določa čas, ko so podatki na vodilu še veljavni po spremembi stanj na naslovnih vhodih.



Slika 2.15 Časovni diagram pri bralni operaciji statičnega RAM-a

Ti podatki so enaki tistim, ki jih srečamo pri ROM-ih. Na sliki 2.16 je prikazan časovni diagram poteka signalov na vhodih in izhodih pri pisalni operaciji statičnega RAM-a. Najpomembnejši časovni intervali, ki jih podajajo proizvalci, so:

- t_{AS} – v tem časovnem intervalu se morajo ustaliti vrednosti na naslovnih vodilih, šele nato se lahko signala CS in WE postavita v aktivno stanje.

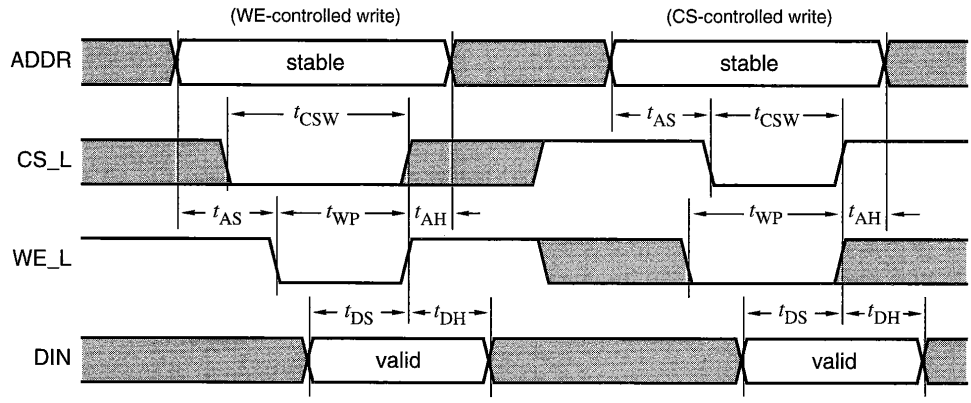
- t_{AH} – je določen podobno kot t_{AS}. Naslovi morajo ostati veljavni še nekaj časa po prehodu signalov CS in WE v neaktivno stanje.

- t_{CSW} – signal CS mora preiti v aktivno stanje vsaj toliko časa prej preden se konča pisalni cikel.

- t_{WP} – pisalni impulz mora trajati vsaj tako dolgo, da se podatki zanesljivo zapišejo v izbrano celico.

- t_{DS} – podatki morajo biti prisotni in stabilni vsaj toliko časa pred koncem pisalnega cikla.

- t_{DH} – podobno kot pri t_{DS} morajo biti podatki na vhodu vsaj še toliko časa pred koncem pisalnega cikla.



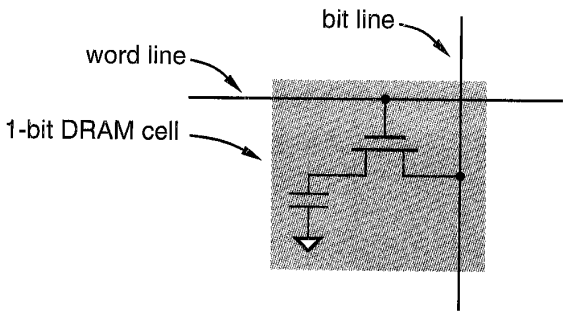
Slika 2.16 Časovni diagram pri pisalni operaciji statičnega RAM-a

2.2.2 Dinamični RAM (DRAM)

Za izvedbo osnovne spominske celice statičnega RAM potrebujemo štiri logična vrata v diskretni izvedbi oz. štiri do šest tranzistorjev pri namensko izdelanem SRAM integriranem vezju. To je ena glavnih omejitev, ko želimo povečati kapaciteto spomina v integriranem vezju. V ta namen so bile razvite spominske celice, pri katerih rabimo za shranitev enega bita le en tranzistor.

2.2.2.1 Zgradba dinamičnega RAM-a.

Iz osnov digitalnih vezij vemo, da ne moremo zgraditi bistabilnega člena z enim tranzistorjem. Namesto uporabe več tranzistorjev se pri dinamičnem RAM-u (DRAM) uporablja princip shranitve električnega naboja v majhnem kondenzatorju s pomočjo MOS tranzistorja. Na sliki 2.17 je prikazana spominska celica enobitnega DRAM-a, ki jo aktiviramo tako, da linijo za izbiro besede priključimo na višji potencial. Za vpis logične enke priključimo bitno linijo na višji potencial in naboj steče skozi tranzistor v kondenzator. Pri pisanju logične nič je bitna linija na nižjem potencialu in naboj odteče iz kondenzatorja preko tranzistorja.

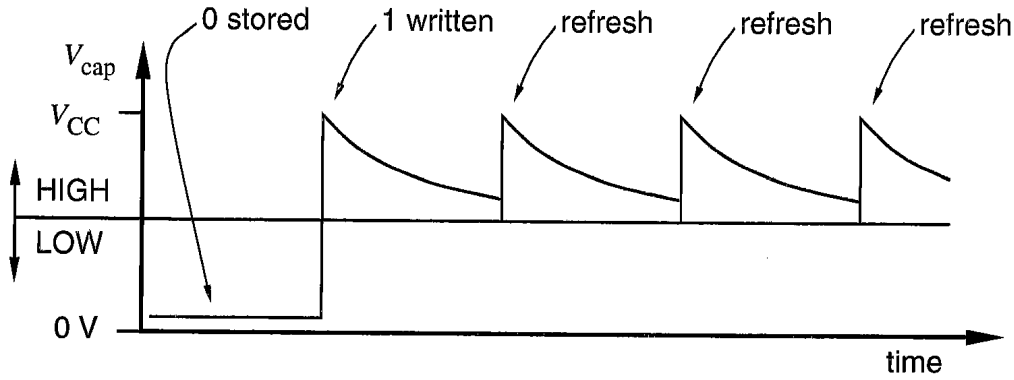


Slika 2.17 Spominska celica dinamičnega RAM-a (DRAM)

Pri branju vsebine celice je treba najprej dvigniti potencial bitne linije na polovico vrednosti potenciala logične ena. Nato se linija za izbiro besede postavi v stanje logične ena. Glede na stanje v spominski celici se v njej napetost nekoliko dvigne (Pri logični nič) oz. nekoliko zniža (pri vpisani logični ena). Dodatno vezje detektira to spremembo in na podlagi tega prebere logično ena oz nič. Pri branju celice se torej napetost celice nekoliko spremeni, zato jo treba zapis celice osvežiti s ponovnim zapisom stare vsebine.

Kapacitivnost kondenzatorja (spominske celice) je zelo majhna vendar je impedanca tranzistorja preko katerega se dostopa do celice zelo visoka vendar končna. To pomeni, da preteče kar nekaj časa od stanja, ko je kondenzator poln, do stanja, ko se napetost tako zniža, da je v njem že logična nič.

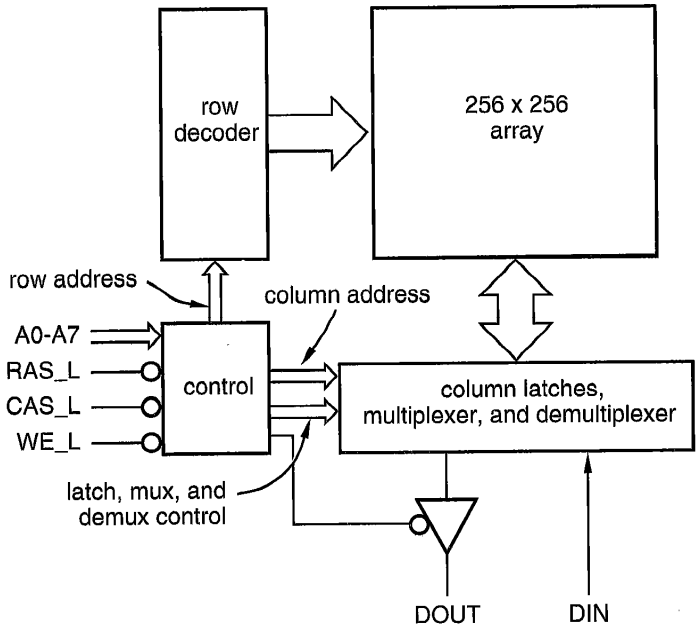
Zaradi tega je treba pri dinamičnem RAM-u osveževati vsebino zapisa v posameznih celicah. Pri prvih RAM-ih je bil cikel osveževanja 4 ms. Pri vsaki osvežitvi se najprej prebere vsebino celice in se jo nato obnovi v izhodiščno stanje. Na sliki 2.18 je prikazan časovni potek napetosti pri vpisu logične ena in postopek nadaljnjega osveževanja vsebine.



Slika 2.18 Potek osveževanja vsebine spominske celice

Današnji dinamični RAM-i že vsebujejo po 256 Mbitov in več. Če bi morali osvežiti vsebino celice vsake štiri milisekunde, potem bi naleteli na težavo, saj bi imeli na razpolago manj kot 1 ns za osvežitev ene celice, ne da bi pri te upoštevali še potreben čas za branje oz. pisanje. Ker pa so sodobni RAM-i zasnovani v obliki dvodimenzionalnega polja, se osvežitev izvede istočasno za celotno vrstico. Prvi DRAM-i so imeli po 256 vrstic. Zato je bilo treba izvesti

256 osvežitev vsake 4 ms oz. eno vsakih 15,6 μs. Novejše izvedbe imajo 4096 vrstic, ki jih je treba osvežiti vsakih 64 ms oz. eno vrstico vsakih 15,6 μs. Vsaka osvežitev traja v povprečju manj kot 100 ns, zato je še vedno na razpolago 99% časa za pisalno-bralne operacije.



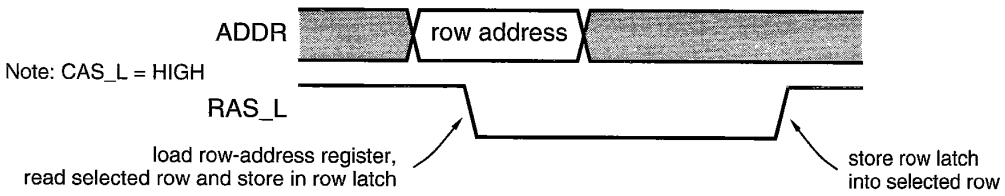
Slika 2.19 Zgradba 64K x 1 DRAM-a

Na sliki 2.19 je prikazana zgradba DRAM-a s kapaciteto 64K x 1. Taka oznaka velja za logično zgradbo, čeprav je fizično to izvedeno v obliki polja velikosti 256 x 256 bitov. Čeprav ima spominska enota 64K lokacij, ima integrirano vezje le osem multipleksiranih naslovnih vhodov. Celoten 16-bitni naslov se vpiše v dveh korakih, ki ju krmilimo s signaloma RAS_L (Raw Address Strobe) in CAS_L (Column Address Strobe). Z multipleksiranjem privarčujemo na številu priključkov integriranega vezja. Naslavljanje se torej izvaja po metodi »dveh korakov«, ki se zelo pogosto uporablja pri DRAM-ih.

Pri večjih DRAM-ih so tudi polja večja ali pa uporabimo več polj. Prednost večjega števila polj je lažje povezovanje pri izdelavi vezja. Poleg tega je pri paralelnih poljih možno izvajati več operacij istočasno. Tako se npr. v enem delu DRAM-a (polju) odvija branje, v drugem pa poteka pisanje.

2.2.2.2 Časovni potek branja in pisanja pri DRAM-u.

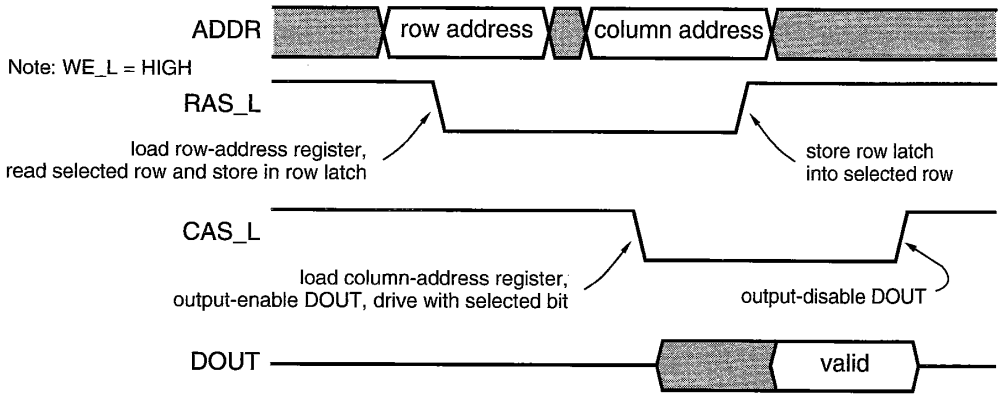
Obstaja več načinov določanja časovnih potekov signalov pri DRAM-ih, ki se razklikujejo glede na izvedbo vezja. Prikazali bomo najpogostejši potek ciklov pri standardnih izvedbah DRAM-ov. Prva presenetljiva novost pri DRAM-u je dejstvo, da ne rabimo takta. Operacije se začenjajo in končujejo na padajočo in na naraščajočo stranico signalov RAS_L in CAS_L. Časovni diagram *osvežitvenega cikla* je (*RAS-only refresh cycle*) je prikazan na sliki 2.20.



Slika 2.20 RAS osvežitveni cikel

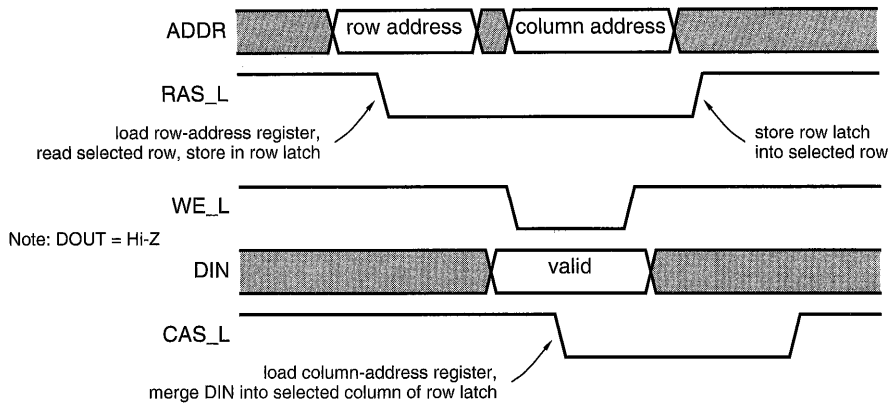
V tem ciklu se osveži izbrana vrstica spomina ne da bi se vpisalo ali prebralo izbrani podatek na zunanjih priključkih spomina. Cikel se začne z naslavljanjem multipleksiranih naslovnih vhodov (8 bitov pri uporabi 64K x 1 DRAM-a) in z aktiviranjem krmilne linije RAS_L. Ob padajoči stranici signala RAS_L shrani DRAM naslov vrstice v notranji register naslovov vrstic, prebere vsebino vrstice in jo zapiše v vrstični zapah. Za osvežitev celotnega spominskega prostora v 64K x 1 DRAM-u je treba ponoviti ta postopek 256. in uporabiti vseh 256 naslovov vrstic v 4 ms. Pri tem lahko uporabimo osembitni števec za naslavljanje posameznih vrstic in časovnik, ki začne z osveževanjem vsakih 15,6 μs.

Bralni cikel, ki je prikazan na sliki 2.21, se začne podobno kot osvežilni cikel z branjem in zapisom izbrane vrstice v interni zapah. V naslednjem koraku se na multipleksiranem naslovnem vhodu postavi naslov stolpca, ki se ob padajoči stranici CAS_L shrani v register naslovov stolpcev. Ta naslov se uporablja za izbiro določenega bita, pred tem prebrane vrstice, ki se nato posreduje na izhodni priključek DRAM-a. Bit je na izhodu (tristanjski izhod) dostopen tako dolgo, dokler je krmilni signal CAS_L v aktivnem stanju. Ko krmilni signal RAS_L preide v neaktivno stanje, se celotna vrstica vpiše nazaj v polje.



Slika 2.21 Časovni diagram bralnega cikla DRAM-a

Pisalni cikel je prikazan na sliki 2.22. Postopek se začne podobno kot osvežilni ali bralni cikel. Krmilni signal WE_L (write enable) je treba aktivirati pred aktiviranjem CAS_L. S tem dosežemo, da je pin DOUT onemogočen (visokoimpedančno stanje) v preostalem delu cikla, čeprav signal CAS_L preide občasno v aktivno stanje. Ko je izbrana vrstica prenešena v vrstični zapah, aktivno stanje WE_L povzroči, da se vhodni bit na DIN vratih prenese v vrstični zapah na mesto, ki je določen z naslovom stolpca. Ko se ob naraščajoči stranici RAS_L vrstica vpiše v polje, že vsebuje novo vrednost v izbranem stolpcu.



Slika 2.22 Časovni diagram pisalnega cikla DRAM-a

Poleg teh so pri standardnih DRAM-ih definirani še nekateri cikli, ki niso prikazani na zadnjih dveh slikah:

V ti. »*CAS-before-RAS refresh*« osvežitvenem ciklu se izvaja osvežitev brez potrebe po predhodnem generiranju naslova vrstice, ki jo izda števec. Če je CAS_L aktiven pred aktiviranjem signala RAS_L, se osveži tista vrstica v DRAM-u, ki jo določa števec. Zatem se stanje števca poveča za ena. Na ta način je možno poenostaviti celotno vezje, saj ne potrebujemo zunanjega števca za osvežitev in tako zmanjšamo število vhodov multipleksorja pri naslavljanju DRAM-a s treh (vrstica, stolpev, osvežitev) na dva.

»*A read-modify-write*« cikel se začne podobno kot bralni cikel in omogoča branje podatkov na izhodu DOUT takrat, ko je krmilni signal CAS_L aktiven. Za zapis podatka nazaj na isto lokacijo lahko signal WE_L aktiviramo naknadno.

»*Page-mode read cycle*« omogoča branje podatkovne vrstice (page) brez ponavljanja celotnega *RAS-CAS* cikla. Ker je v vrstičnem zapahu že shranjena celotna vrstica, je treba večkrat postaviti krmilni signal CAS_L v nizko stanje, medtem ko krmilni signal ostaja ves čas aktiven. Ob vsaki padajoči stranici krmilnega signala CAS_L se sprosti nov naslov in na izhodu DOUT se pojavi nova vsebina izbrane lokacije. Z uporabo tega cikla je možen hitrejši dostop do spominskih lokacij, ki so si prostorsko blizu. Pri mikroprocesorjih se to zelo pogosto uporablja npr. pri branju ukazov oz. pri nalaganju »cache« spomina.

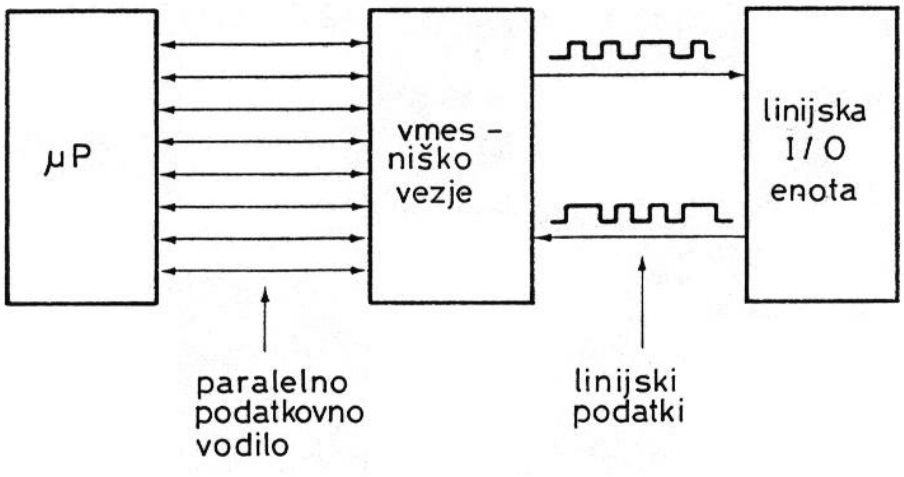
3. Serijski prenos podatkov

3.1 Asinhronski serijski prenos podatkovi

Pri asinhronskem prenosu lahko prenosna naprava pošlje podatke k sprejemniku, v katerem koli času brez potrebne sinhronizacije s sprejemnikom. Na sliki 3.1 je prikazana blokovna shema takšnega prenosnega sistema. Na sliki sta prikazani dve osnovni operaciji vmesniškega vezja:

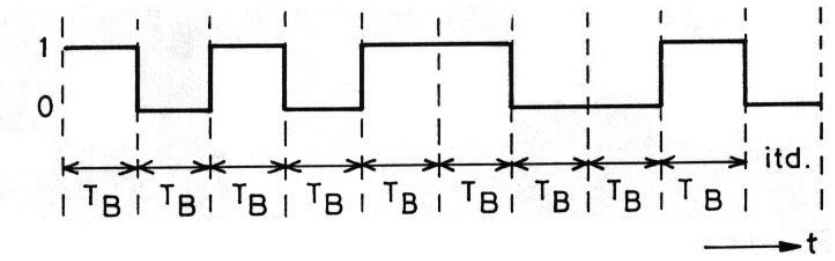
1. Sprejme 8-bitno besedo paralelno iz mikroprocesorskega podatkovnega vodila in jo pretvori v 8-bitno besedo, ki jo pošlje v serijsko napravo.
2. Sprejme serijski podatkovni signal iz serijske naprave in ga pretvori v 8-bitno paralelno besedo. To potem preko podatkovnega vodila posreduje mikroprocesorju.

Glavna funkcija vmesniškega vezja je pretvarjanje paralelnih podatkov v serijske in obratno. Predno si to podrobneje razložimo, si še enkrat poglejmo kako izgleda serijski podatkovni signal.



Slika 3.1: Serijska povezava mikroprocesorja z vhodno/izhodno enoto

Serijski podatkovni signal je razdeljen na časovne intervale, ki jih imenujemo bitni časi (Slika 3.2). V bitnem časovnem intervalu (T_B) je lahko nivo signala 0 ali 1, spreminja pa se lahko le na začetku vsakega bitnega časovnega intervala.

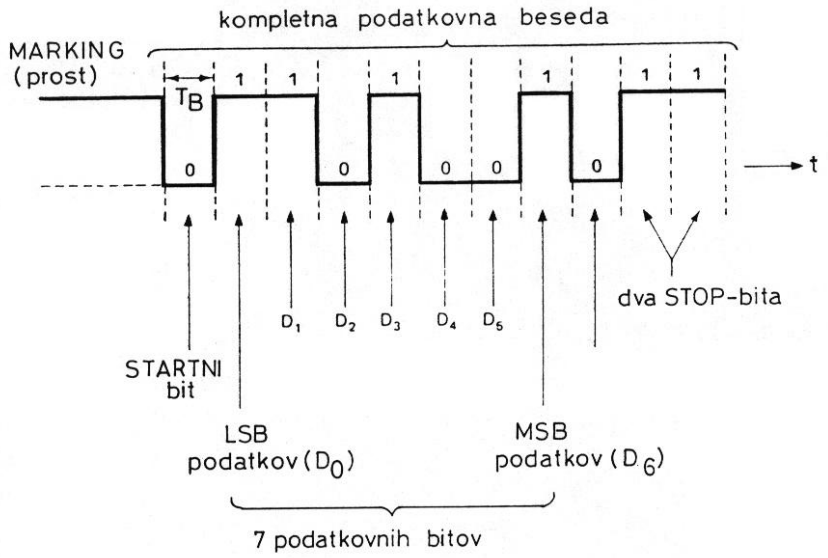


Slika 3.2: Serijski prenos podatkovne besede.

Če se asinhronski serijski podatki prenašajo med dvema napravama, je za prenos ene podatkovne besede uporabljen standardni format, ki je sestavljen iz treh (ali štirih) delov:

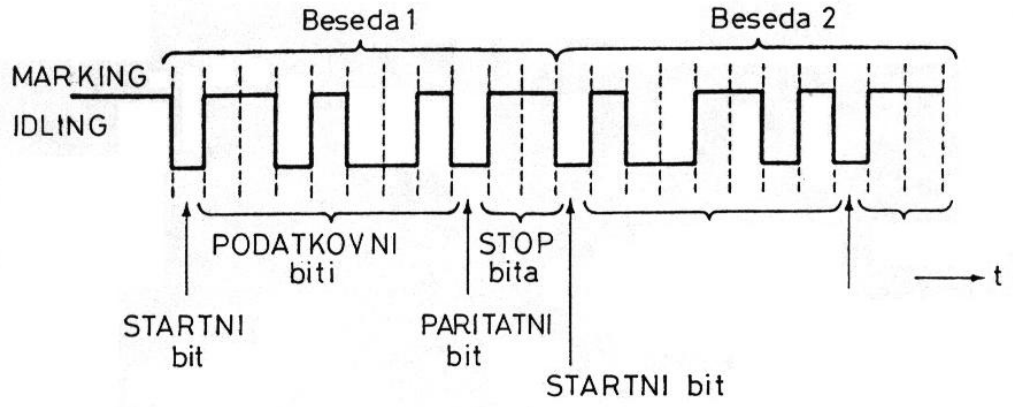
1. Startni bit, ki je vedno 0.
2. 5 do 8 podatkovnih bitov, ki prenašajo informacijo.
3. Dodatni paritetni bit za ugotavljanje napak. Če je vključen paritetni bit, je lahko uporabljena soda ali liha pariteta.
4. 1; 1,5 ali 2 STOP bita, ki sta vedno 1. Najpogosteje sta 2 STOP bita.

Za nek sistem je število podatkovnih bitov in število STOP bitov določeno že s konfiguracijo vezja. Na sliki 1.3 je prikazan primer serijske podatkovne besede, ki uporablja 7 podatkovnih bitov, en sodi paritetni bit in 2 STOP bita. Ta format je uporabljen npr. pri prenosu ASCII kode.
Kompletna serijska podatkovna beseda na sliki 3.3 se začne s START bitom, ki je vedno 0. Pred startnim bitom je na signalni liniji stalno visok logični nivo. To se imenuje »marking« ali »idling« (prazen, prost). Kadarkoli ni prenašanja podatkov, je linija vedno v tem stanju. Prenos neke podatkovne besede se začne označen pri prehodu signala od 1 na 0 ob nastopu START bita. Temu sledi 7 podatkovnih bitov z začetnim LSB in končnim MSB bitom. V našem primeru je aktualen prenešen podatek 1001011, kar pomeni v ASCII kodi črko K. Podatkovnim bitom sledi sodi paritetni bit, v tem primeru je 0, ker ima sedem podatkovnih bitov sodo število enic. Paritetnemu bitu sledita dva STOP bita, ki sta vedno 1.



Slika 3.3: Dodatni biti pri serijskem prenosu

Asinhronski prenos podatkov se pogosto uporablja za prenos podatkov iz ene naprave v drugo. Na sliki 3.4 je prikazan serijski signal za prenos dveh zaporednih besed. Če so serijske besede prenešene druga za drugo, START bit naslednje besede neposredno sledi dvema STOP bitoma predhodne besede. Iz tega vidimo, kakšno število besed v določenem času lahko prenašamo. Če pa se besede ne prenašajo z maksimalno možno gostoto, tedaj je vedno od zadnjih dveh STOP bitov pa do naslednjega START bita na podatkovni liniji visoko logično stanje (idling).



Slika 3.4: Serijski prenos dveh besed brez presledka

Začetna sinhronizacija naprave, ki želi sprejeti serijski podatkovni signal, je izvedena z negativnim robom START bita prve besede. To pomeni, da je naslednjih 10 prenešenih bitov, 7 podatkovnih, paritetni bit in 2 STOP bita. Po sprejemu STOP bitov sprejemna naprava čaka na naslednji startni signal tj. na negativni rob startnega signala naslednje besede. Na ta način so START in STOP biti uporabljeni za sinhronizacijo sprejemnika in oddajnika serijskih podatkov.

3.1.1 Vloga paritetnega bita

Kljub vedno bolj zanesljivi opremi lahko pride pri prenosu podatkov do napak. Tudi če pride do napake zelo malokrat, pa so prenešeni podatki lahko popolnoma neuporabni. Pri postopkih kodiranja podatkov obstaja veliko metod za odkrivanje napak, ki se razlikujejo glede učinkovitosti in zapletenosti izvedbe.
Ena najpogosteje uporabljenih metod za odkrivanje napak je ti. pariteta (ang. parity). Skupini bitov, ki tvorijo nek podatek, dodamo ti. paritetni bit. Njegova vrednost je ena ali nič glede na število enk oziroma ničel v podatkovnem nizu.
Metoda se uporablja v dveh različicah. Pri ti. sodi pariteti je bit izbran v primeru, ko je število logičnih enk v podatkovnem nizu skupaj s paritetnim bitom soda vrednost. Če predpostavimo, da imamo podatkovno besedo 1000011, ki ponazarja ASCII znak C. Ta beseda vsebuje tri logične enke. Da bi pri prenosu imeli sodo število logičnih enk, bo paritetni bit v stanju logične enke. Celotna beseda, ki jo želimo prenesti je sestavljena iz paritenega bita in iz podatka:

1 1 0 0 0 0 1 1

paritetni bit

Če je sodo število logičnih enk v podatku, je paritetni bit v stanju logične nič. Tako bo pri podatkovni besedi 1000001 (ASCII znak A), ki ima sodo število logičnih enk, paritetni bit v stanju logične nič (**0**1000001).

Liha pariteta se uporablja na enak način, le vrednost paritetnega bita se določa tako, da je v podatkovni besedi vključno s paritetnim bitom liho število enk. Tako je npr. pri podatkovni

besedi 1000001 paritetni bit v stanju logične ena, pri podatkovni besedi 1000011 pa je paritetni bit 0.
Paritetni bit postane del podatkovne besede, ki jo prenašamo. Če prenašamo npr. 7-bitne ASCII znake, je tako podatkovna beseda z dodanim paritetnim bitom 8-bitna.
Omenjena metoda omogoča odkrivanje napake pri prenosu podatkov oz. znakov, ne omogoča pa detektiranja napačnega bita v posamezni besedi. Obstajajo pa tudi metode, ki omogočajo detektiranje in tudi popravljanje okvarjenega bita v preneseni besedi.

3.1.2 Hitrost prenosa

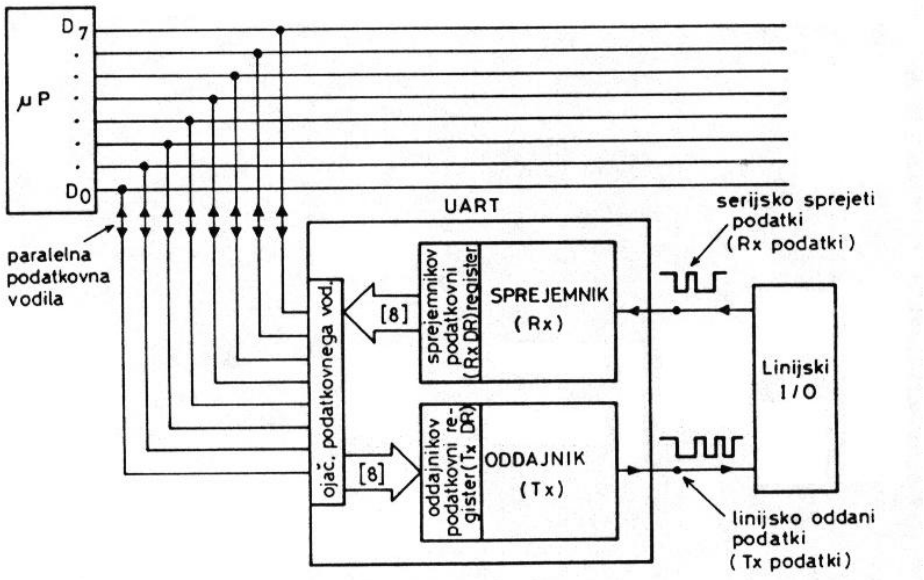
Hitrost prenosa v Baud-ih je število bitov informacije, ki se prenese v eni sekundi.

$$Baud\ rate = \frac{1}{T_B} \quad (bit/s)$$

Za pravilno interpretacijo podatkov mora biti hitrost prenosa za sprejemnik in oddajnik enaka.

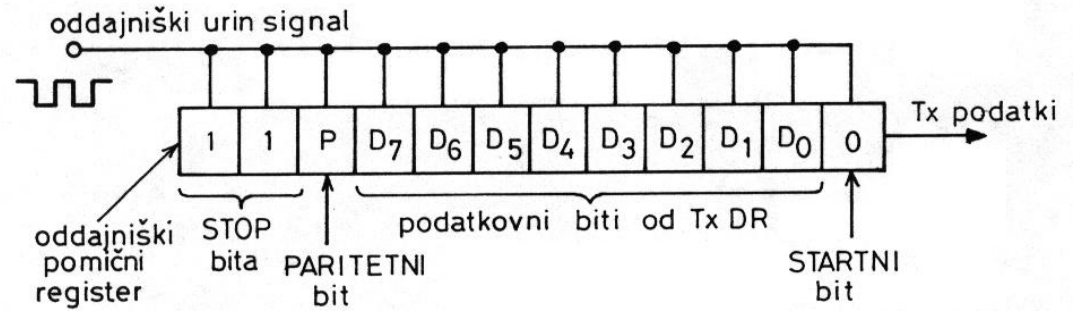
3.2 Paralelno/serijski vmesnik (UART)

Iz opisanega delovanja serijskega prenosa je razvidno, da potrebujemo vmesniško vezje za pretvorbo paralelnih in serijskih podatkov (Slika 3.5). Pretvornik imenujemo UART (Universal Asynchronous Receiver Transmitter). Vmesnih vsebuje serijski sprejemnik (Rx), ki ima serijski vhod in ga pretvori v paralelni format, ki se shrani v sprejemnikov podatkovni register (RxDR) za eventualni prenos v mikroprocesor. Serijski oddajnik (Tx) dobi paralelno podatkovno besedo iz oddajnikovega podatkovnega registra (TxDR) in jo pretvori v serijski format za prenos. Dvosmerni ojačevalnik podatkovnega vodila posreduje paralelne podatke iz mikroprocesorja v TxDR ali iz RxDR k mikroprocesorju preko sistemskega podatkovnega vodila.



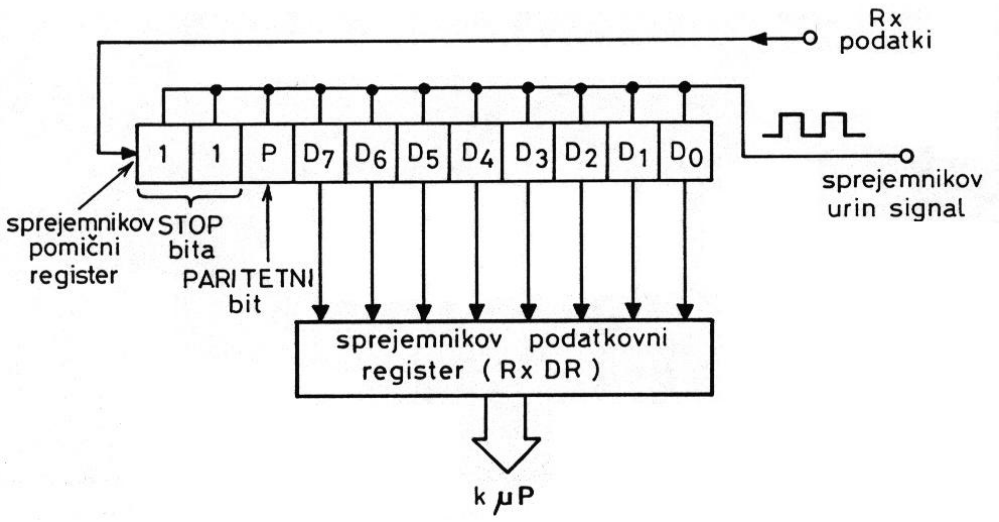
Slika 3.5: Osnovna zgradba UART

Če želi mikroprocesor prenesti podatkovno besedo v serijsko izhodno enoto, pošlje podatkovno besedo v UART-ov oddajniški podatkovni register (TxDR). Krmilna logika Tx registra vzame to podatkovno besedo in ji doda START bit, paritetni bit in potrebno število STOP bitov. Kompletna podatkovna beseda se potem vstavi v pomični register – oddajniški pomični register (Slika 3.6).



Slika 3.6: Oddajniški pomični register

Vsebina tega registra se pomika desno s hitrostjo, ki jo določa urin signal TDC (Transmit Data Clock). Tako se formira serijska podatkovna beseda (TxData), ki se potem prenese v izhodno enoto. Urin signal za prenos podatkov mora imeti enako frekvenco kot je zahtevana hitrost prenosa (Baud rate).
Če želi serijska zunanja enota poslati podatke mikroprocesorju, potem prenese serijsko podatkovno besedo na UART-ov sprejemnik skozi RxData vhod. Če krmilna logika v Rx sekciji pošlje negativni rob na RxData linijo, se ta interpretira kot START bit. Ta potem premakne ostali del serijske podatkovne besede v pomikalni register sprejemnika (Slika 3.7) s hitrostjo sprejemnikove podatkovne ure (RECEIVER DATA CLOCK). Ko je kompletna beseda v pomičnem registru, se podatkovni del (D_7 do D_0) paralelno premakne v RxDR. Če je potrebno, lahko sedaj mikroprocesor prebere vsebino RxDR na enak način kot spominsko lokacijo.

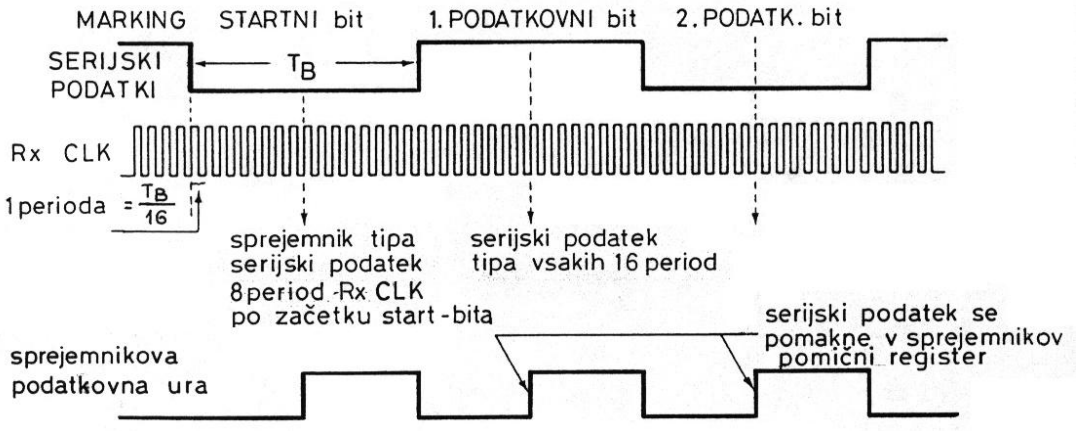


Slika 3.7: Pomični register sprejemnika

Pri delovanju sistema za asinhronski serijski prenos mora bit poskrbljeno za uglašenost med UART in serijsko vhodno/izhodno enoto, kakor tudi za pravilen format serijskega podatka (število podatkovnih bitov, pariteta, in število STOP bitov). Ta informacija je običajno programirana v UART preko mikroprocesorja. UART ima interni krmilni register v katerega lahko mikrokrmilnik piše tako kot v spominske lokacije. Mikroprocesorjeve instrukcije omogočajo pošiljanje 8 bitne kode preko podatkovnega vodila v krmilni register.

Kot smo že omenili, se sprejemnik na začetku sinhronizira z negativnim robom startnega bita. To pomeni, da sledijo podatkovni, paritetni in stop biti. Da bi olajšali sinhronizacijo s serijskimi podatki, uporablja UART zunanji urin signal, ki ima za faktor 16 višjo frekvenco od hitrosti prenosa pulza (baud rate). Če je npr. hitrost prenosa pulzov 110, mora imeti zunanji urin signal frekvenco 1760 Hz. Ta signal je dostopen na UART-ovem RxCLK priključku. Ena perioda je torej 1/16 bitnega čas T_B.

Ko sprejemnik začuti prvi negativni rob startnega bita, čaka 8 period RxCLK signala in nato preizkusi stanje na serijskem vhodu (Slika 3.8), če je res v nizkem logičnem stanju.



Slika 3.8: Testiranje logičnega stanja na vodilu s pomočjo internega urinega signala

Ta pove sprejemniku, da je na podatkovni liniji START bit. Vzorčenje serijskega vhoda se opravi na polovici intervala START bita, tj. po osmih RxCLK pulzih. Po ugotovitvi startnega bita, sprejemnik vzorči serijske podatke v intervalih po 16 RxCLK period, kar zagotavlja, da je vsak podatek preizkušen na sredini T_B. Vsak od teh vzorcev je potem premaknjen v sprejemnikov pomikalni register ob naraščajočem robu sprejemnikove podatkovne ure, ki je 1/16 RxCLK signala. Sprejemnikova podatkovna ura je enaka hitrosti prenosa pulzov (Baud rate).
Ko je enkrat cel podatek v sprejemnikovem podatkovnem registru, vezje v UART avtomatično kontrolira, če so vsi STOP biti 1. Če je kateri od STOP bitov enak 0, se aktivira zastavični bit v UART-ovem internem statusnem registru. Tega lahko bere mikroprocesor tako kot spominsko lokacijo. Mikroprocesor lahko med sprejemom teh podatkov ugotavlja stanje UART-ovega statusnega bita in ob eventualni napaki ustrezno ukrepa.

Na enak način deluje tudi kontrola paritete. V tem primeru je v igri paritetni zastavični bit, ki je prav tako del statusnega registra UART.

Kot smo videli na sliki 3.4, se podatkovni biti prenašajo v RxDR paralelno, kjer so na razpolago mikroprocesorju. Ko je prenos v RxDR končan, sprejemnik takoj začne »iskati«

novo besedo in jo vpiše v pomični register. Slednja počaka na prenos v RxDR, dokler mikroprocesor ne prebere prejšnje besede v RxDR. Tako se lahko zgodi, da pomični register sprejme že tretjo besedo, preden se je druga vpisala. Tako se druga beseda izgubi. Če se to zgodi, UART aktivira »Over Run Error Flag«. Mikroprocesor mora torej prebrati podatke iz RxDR vsaj v času, ko se nova podatkovna beseda kompletira v sprejemniku.

3.3 RS-232 standard za serijski prenos podatkov

EIA združenje (Electronics Industry Association) je privzelo serijski komunikacijski standard RS-232-C. Standard določa napetostne nivoje signalov in ti. usklajevalne signale med podatkovnimi terminali.

RS-232-C napetostni nivoji so:

Logična 1 (mark) = < - 3 V
Logična 0 (space) = > + 3 V

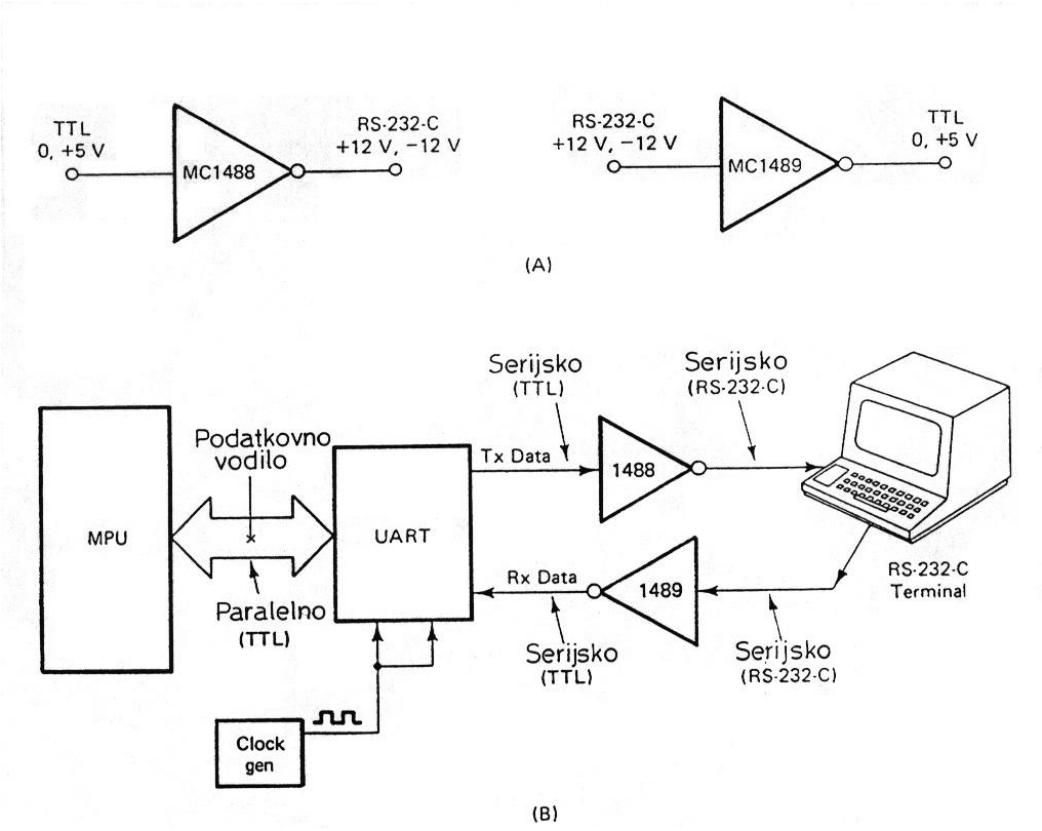
Katerakoli napetost med -3 V in + 3 V ni razpoznavna. Tipična napetost sistema RS-232-C je -12 V in + 12 V za 1 oziroma za 0.

Mnoge standardne komponente uporabljajo TTL napetostne nivoje (0, 5 V) in zato niso neposredno kompatibilne z zunanjimi enotami, ki uporabljajo RS-232-C. Za to potrebujemo vmesniška vezja za pretvorbo med TTL in RS-232-C napetostnimi nivoji. Na sliki 1.9a vidimo dve standardni integrirani vezji za pretvorbo napetostnih nivojev. Vezje 1488 pretvarja TTL vhode v RS-232-C izhode, vezje 1489 pa pretvarja RS-232-C napetostne vhode v TTL izhode.

Na sliki 3.9b vidimo primer vmesnika med mikroprocesorjem, ki uporablja TTL nivoje in terminalom, ki ima RS-232-C napetostne nivoje. Mikroprocesorska enota komunicira z RS-232-C video terminalom, ki uporablja serijske podatke z napetostnimi nivoji RS-232-C. Če mikroprocesor pošilja podatke v terminal, se to odvija v nekaj korakih:

1. Mikroprocesor pošlje v UART paralelne podatke, ki imajo TTL napetostne nivoje.
2. UART pretvori paralelne podatke v serijske in jih pošlje na podatkovni izhod TxData.
3. Serijski podatki se na int. vezju 1488 pretvorijo na napetostne nivoje RS-232-C.

Če potuje podatek od terminala k mikroprocesorju, se proces odvija v obratni smeri.



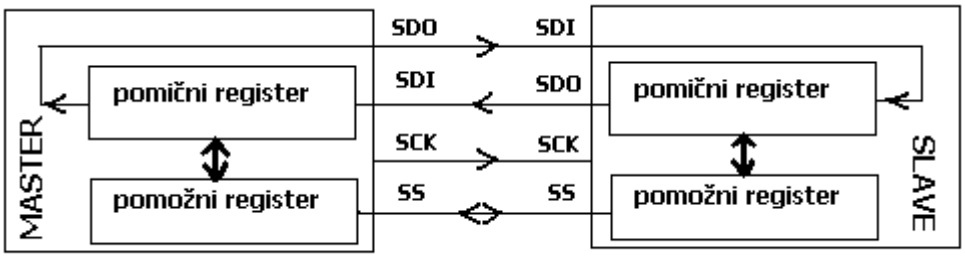
Slika 3.9: Pretvorba napetostnih nivojev pri serijskem prenosu podatkov

3.4 SPI – sinhronski serijski periferni vmesnik (Synchronous Serial Peripheral Interface)

Serijski periferni komunikacijski vmesnik je namenjen povezovanju mikrokrmilnika s perifernimi enotamic (Slika 3.10a). Poleg tega omogoča komunikacijo v večprocesorskem sistemu (torej tam, kjer je več »master« enot). Med perifernimi enotami so lahko preprosti pomični registri ali pa tudi zahtevnejše kot npr. LCD krmilniki, A/D pretvorniki ipd.
Vmesnik lahko deluje v vlogi »master« ali »slave«. Pri delovanju vmesnika v vlogi »master« lahko doseže hitrost 1 Mbit/s, pri delovanju v vlogi »slave« pa znaša 2 Mbit/s.

Vmesnik ima štiri vhodno/izhodne linije:

- SDI: serijski digitalni vhod (serial digital input)
- SDO: serijski digitalni izhod (serial digital output)
- SCK: urin signal (serial clock)
- SS: krmilni signal v »slave« režimu delovanja (slave serial)



Slika 3.10a: Povezave »master« in »slave« enot z SPI vmesnikom

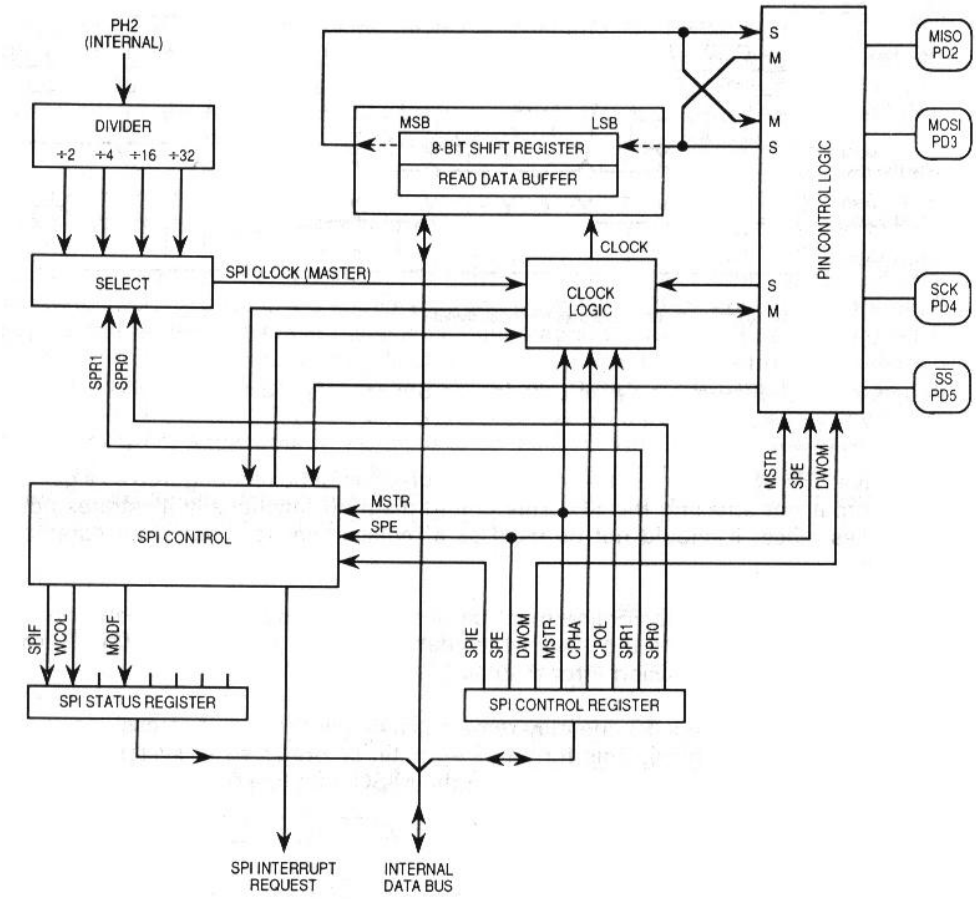
Pri delovanju v vlogi »master« se programsko določi hitrost prenosa (bit rate) z nastavitvijo frekvence osnovne ure. Krmilno vezje skrbi tudi za odkrivanje napak pri prenosu. Tako preprečuje vpisovanje podatkov v serijski pomični register v času, ko se izvaja prenos. Poleg tega onemogoča simultano pošiljanje podatkov več kot ene »master« enote v primeru, ko imamo v sistemu več »master« enot.

SPI prenos se odvija istočasno v obeh smereh. Pomik in vzorčenje podatkov se odvija na obeh podatkovnih linijah s pomočjo osnovnega urinega signala.

Na sliki 3.10b je prikazana blokovna shema SPI vmesnika pri Motorolinem mikrokrmilniku MC68HC11. Med prenašanjem podatkov se lahko hkrati oddaja 8-bitni podatek na izhodnem priključku in sprejema 8-bitni podatek na vhodnem priključku.
To je izvedeno tako, da 8-bitni pomični register v »master« enoti in 8-bitni pomični register »slave« tvorita ti. krožni pomični register. Pri prenosu, se vsebina v tem registru pomakne za 8 bitov. Pri tem se njuni vsebini zamenjata.

Najpomebnejši element vmesnika je blok, ki vsebuje pomični register in podatkovni »buffer«. Imamo en »buffer« v smeri oddaje podatka in dva »bufferja« v smeri sprejema podatkov. To pomeni, da ni možno vpisovati novega podatka v izhodni »buffer«, dokler predhodni ni bil v celoti poslan. Pri sprejemu pa se podatek prenaša v paralelni »buffer« in pomični register lahko nemoteno sprejema naslednji niz bitov. Treba je le zagotoviti, da se pravočasno prebere podatek iz sprejemnega »bufferja« preden je naslednji serijski podatek pripravljen za prenos. Za branje podatka iz sprejemnega »bufferja« in za pošiljanje podatka v pomični register uporabimo isti naslov.

Vmesnik ima štiri vhodno/izhodne priključke. SCK je izhod, če SPI vmesnik deluje kot »master« oz vhod, če deluje kot »slave«. Če deluje kot »master«, potem se na CLK izhodu avtomatsko generira 8 urinih impulzov, ki jih posreduje blok »Clock Logic«. Pri delovanju v funkciji »slave« je SCK vhod, urin signal iz »master« enote pa sinhronizira prenos podatkov med »master« in »slave« enoto. V obeh načinih delovanja se podatki pomikajo pri eni stranici urinega signala, vzorčijo pa se pri drugi stranici urinega signala, ko je le ta že stabilen. Na katero strmino se odvija pošiljanje oz vzorčenje, se določi programsko s pomočjo konfiguracijskega registra.



Slika 3.10b: SPI vmesnik pri mikrokrmilniku MC68HC11

MISO in MOSI podatkovna priključka omogočata pošiljanje in sprejemanje serijskih podatkov. Če deluje vmesnik kot »master«, potem je MISO podatkovni vhod, MOSI pa je podatkovni izhod. Pri delovanju v »slave« režimu se njuni vlogi zamenjata. V sistemu z več »master« enotami se vse SCK linije združijo kakor tudi vse MOSI oziroma MISO linije.
Če imamo sistem z eno »master« enoto, potem vse ostale enote delujejo kot »slave«. V tem primeru »master«, z izhodoma SCK in MOSI, krmili SCK in MOSI priključke drugih enot. Izbrana »slave« enota pa pošilja podatke preko svojega MISO izhoda na MISO priključek (vhod) »master« enote.

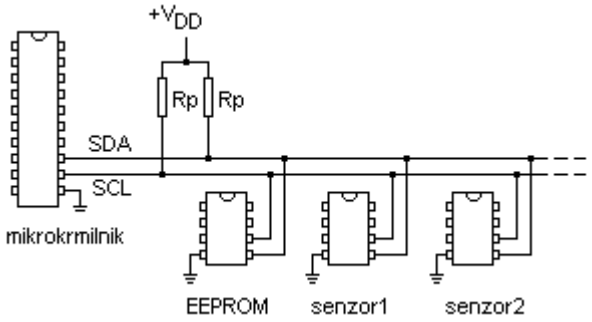
$\overline{SS}$ priključek ima drugačno funkcijo pri delovanju enote v funkciji »master« oziroma »slave«. Pri delovanju v režimu »slave« se uporablja za omogočanje pošiljanja podatkov. Če je $\overline{SS}$ v neaktivnem stanju (visoko), potem enota ne upošteva urinega signala SCK in drži izhodno linijo MISO v visokoimpedančnem stanju. Pri »master« enoti pa $\overline{SS}$ priključek lahko služi kot vhod za detekcijo napake ali pa kot navadni digitalni izhod, ki ni povezan z SPI vmesnikom.

3.5 I^2C protokol za prenos podatkov

I^2C (Inter Integrated Circuit) serijski komunikacijski protokol je bil razvit v osemdesetih letih pri družbi Philips Semiconductors, za prenos podatkov med elektronskimi komponentami na nivoju tiskanega vezja. Razvit je bil z namenom zmanjšanja števila prenosnih linij. Prenos podatkov poteka po dveh linijah in masi. Na eni liniji se nahajajo podatki, na drugi pa taktni signal. Hitrosti prenosa so standardizirane na 100 kbit/s, 400 kbit/s ali 3,4 Mbit/s.

Prenos podatkov po I^2C protokolu so najprej uporabljali pri Philipsovih radijskih in televizijskih sprejemnikih, za nastavljanje ostrine slike in glasnosti. Danes se protokol uporablja v računalništvu, industrijski avtomatizaciji, medicinski opremi, zabavni elektroniki itd.

I^2C je serijski protokol za prenos podatkov med raznimi digitalnimi elektronskimi komponentami. Poteka po dveh povezovalnih linijah in masi. Podatki se nahajajo na liniji SDA (Serial DAta), linija SCL (Serial CLock) pa služi za taktni signal (Slika 3.11).



Slika 3.11: Priklop naprav na I^2C vodilo

Po načinu delovanja je posamezna naprava lahko nadrejena enota (angl. master) ali podrejena enota (angl. slave). Nadrejena enota začne prenos in upravlja s podrejenimi enotami. Nadrejene enote so npr. mikrokrmilniki, podrejene enote pa razni senzorji, prikazovalniki, pomnilniki itd.
SDA in SCL liniji sta priključeni na napajanje preko tokovnega vira ali dvižnih uporov (Slika 3.11). Kadar vodilo ni aktivno, sta obe liniji na visokem logičnem nivoju.
Število priključenih naprav je omejeno le z maksimalno kapacitivnostjo vodila, ki znaša 400 pF.

Značilen potek signalov pri I^2C protokolu:

- Komunikacija po I^2C protokolu se začne tako, da nadrejena enota pošlje na vodilo znak za začetek prenosa, oz. generira START pogoj. Vse podrejene enote čakajo na podatke.
- Nadrejena enota pošlje naslov podrejene enote (7 ali 10 - bitni).
- Nadrejena enota pošlje R/W bit (READ / WRITE). Če je R/W bit '0', bo nadrejena enota pošiljala podatke podrejeni enoti. Če je R/W bit '1', bo nadrejena enota brala podatke s podrejene enote.
- podrejena enota, ki je prepoznala naslov, pošlje potrditev. Ostale podrejene enote čakajo na naslednji START pogoj.

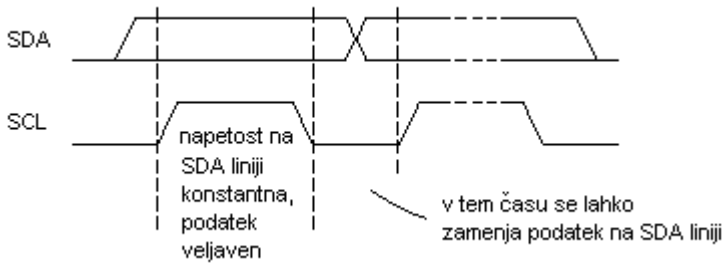
- Nadrejena enota pošilja ali bere podatke po bajtih, po vsakem bajtu podrejena enota pošlje potrditev.
- Nadrejena enota zaključi prenos tako, da generira STOP pogoj.

Tabela 3.1: Standardne hitrosti prenosa podatkov

Režim	**Hitrost prenosa**	**Oznaka**
Standardni režim (angl. standard-mode)	100 kbit/s	F/S (Fast/Standard)
Hitri režim (angl. fast-mode)	400 kbit/s	F/S
Visoko-hitrostni režim (angl. high-speed mode)	3.4 Mbit/s	Hs

3.5.1 Značilnosti I²C protokola na strojnem nivoju

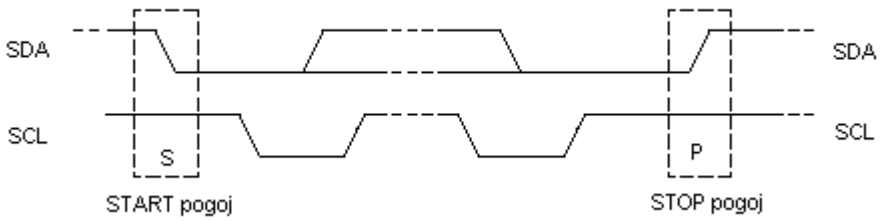
Naprave za priklop na I²C vodilo so izdelane v različnih tehnologijah (CMOS, NMOS, bipolarna...), zato napetostni nivoji za nizek logični nivo ('0') in visok logični nivo ('1') niso absolutno določeni in niso vezani na napajalno napetost. Visok logični nivo je lahko od 0,7 U_{DD} do U_{DD} + 0,5 V, nizek logični nivo pa od – 0,5 V do 0,3 U_{DD}, pri čemer je U_{DD} napajalna napetost.
Za vsak prenešeni bit se generira en taktni pulz na SCL liniji. Ko je SCL signal na visokem logičnem nivoju, se podatek na SDA liniji ne sme spreminjati. Takrat sprejemniki preberejo podatek iz SDA linije. Ko je SCL na nizkem logičnem nivoju, se na SDA pošlje naslednji bit (Slika 3.12).



Slika 3.12: Pravila prenosa podatkov na bitnem nivoju

1.5.1.1 START in STOP pogoja

Vsak prenos podatkov se začne s START pogojem in konča s STOP pogojem (Slika 3.13).

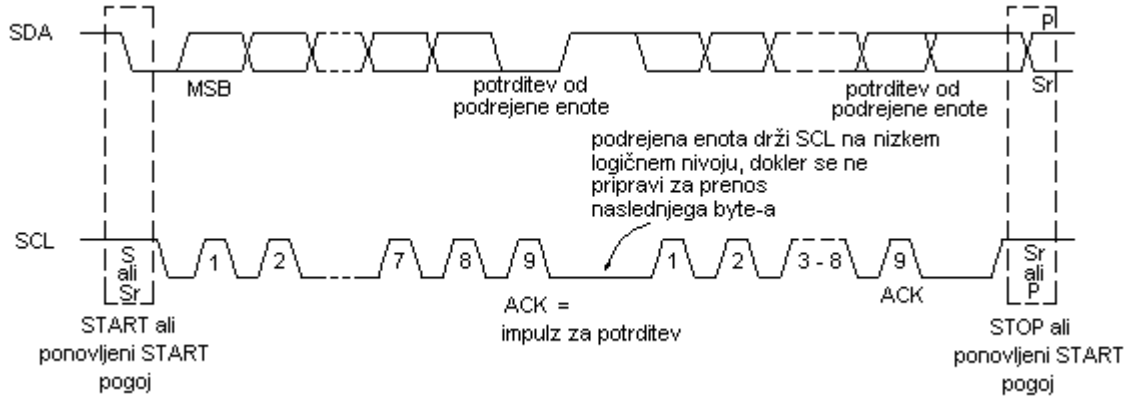


Slika 3.13: START in STOP pogoja

START pogoj (S): SCL je na visokem logičnem nivoju, SDA preide iz visokega na nizek logični nivo. STOP pogoj (P): SCL je na visokem logičnem nivoju, SDA preide iz nizkega na visok logični nivo. START in STOP pogoj vedno generira nadrejena enota. Po vsakem START pogoju ima vodilo status 'zasedeno', po vsakem STOP pogoju pa status 'prosto'.
START pogoj se lahko med prenosom tudi ponovi. Kadar nadrejena enota zaključi komunikacijo z eno napravo in takoj začne z drugo, ne pošlje STOP in START pogoja ampak samo ponovi START pogoj. Vodilo ohrani status 'zasedeno'. To je ponovljen START pogoj (Sr), ki je identičen START pogoju (S).
Detekcija START ali STOP pogoja je enostavna, če imajo naprave vgrajen I^2C I/O priključek. Mikrokrmilniki, ki tega nimajo, morajo vzorčiti SDA linijo najmanj dvakrat na eno SCL periodo.
Pri vsakem START pogoju morajo vse I^2C kompatibilne naprave na vodilu inicializirati svoje delovanje, tudi če je bil poslan med potekom komunikacije.

1.5.1.2 Pravila za prenos bajta podatkov

Vsak bajt mora biti dolg 8 bitov. Število bajtov pri enem prenosu ni omejeno. Vsakemu prenesenemu bajtu sledi potrditev. Prenos bajta se začne z najbolj pomembnim (MSB) bitom (Slika 3.14). Če podrejena enota po prejemu bajta ne more takoj prejeti naslednjega (npr. zaradi procesiranja neke prekinitve), lahko zadrži SCL linijo na nizkem logičnem nivoju in s tem zadrži prenos. Prenos se nadaljuje, ko podrejena enota sprosti SCL linijo. S tem lahko podrejena enota zniža frekvenco do poljubne vrednosti.

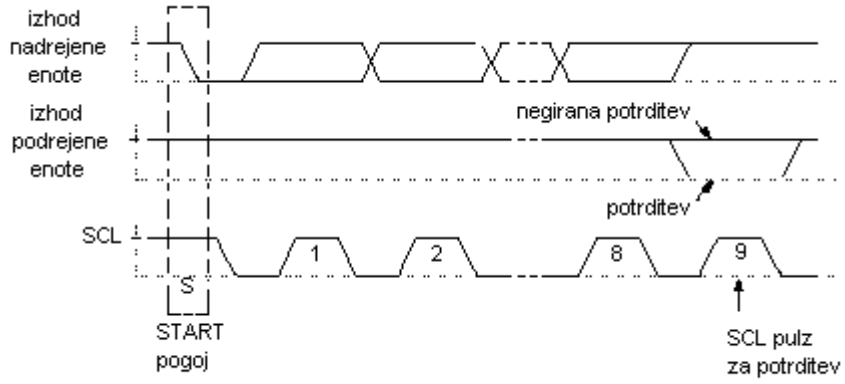


Slika 3.14: Prenos dveh bajtov podatkov

V nekaterih primerih je dovoljen drugačen potek, npr. pri napravah, ki so grajene za CBUS protokol.

1.5.1.3 Potrditev uspešnega sprejema

Po prenosu osmih bitov nadrejena enota generira še en SCL pulz. Sinhrono s tem pulzom sprejemnik generira potrditev tako, da na SDA pošlje impulz nizkega logičnega nivoja (Slika 3.15). Upošteva se tudi določeno časovno trajanje impulzov.



Slika 3.15: Generiranje potrditve

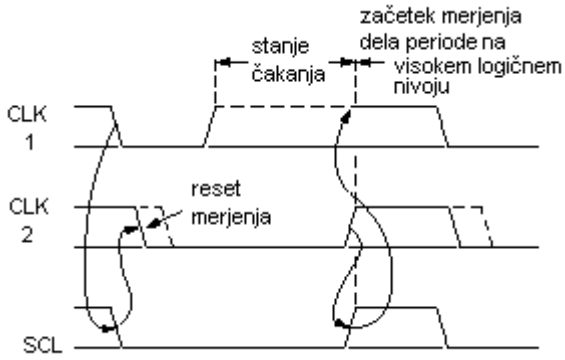
Sprejemnik, ki je bil naslovljen, mora poslati potrditev po vsakem prejetem bajtu, razen v primeru CBUS protokola.
V primeru, da naslovljena podrejena enota ne pošlje potrditve, nadrejena enota pošlje STOP pogoj in začne nov prenos, ali ponovljen START pogoj in ponovi prenos.
Če med prenosom podrejena enota ne more več sprejemati podatkov, po zadnjem sprejetem bajtu ne pošlje potrditve sprejema in nadrejena enota pošlje STOP pogoj ali ponovljen START pogoj.
V primeru, da nadrejena enota sprejema podatke od podrejene enote, nadrejena enota pošilja potrditev za vsak prejeti bajt. Po zadnjem prejetem bajtu ne pošlje potrditve. Podrejena enota takrat sprosti SDA linijo, nadrejena enota pa generira STOP ali ponovljen START pogoj.

3.5.2 ARBITRAŽNA PROCEDURA IN SINHRONIZACIJA SCL SIGNALA

Izhodne stopnje naprav, priključenih na vodilo, morajo biti tipa 'open-drain' ali 'open-collector'. S tem je omogočena izvedba logične IN povezave med logičnim nivojem, ki ga hoče generirati nadrejena enota in dejanskim logičnim nivojem na liniji. Če npr. nadrejena enota 1 pošlje na linijo nizek logični nivo, nadrejena enota 2 pa visok logični nivo, bo linija ostala na nizkem logičnem nivoju. To je izkoriščeno pri sinhronizaciji SCL signala in pri arbitražni proceduri, ki v primeru, da več nadrejenih enot hkrati začne prenos, dovoli nadaljevati le eni.

3.5.2.1 Sinhronizacija SCL signala med vsemi napravami na vodilu

Vsaka nadrejena enota generira svoj taktni signal SCL. Podatek na SDA liniji je veljaven, ko je SCL linija na visokem logičnem nivoju.
Ko nadrejena enota SCL linijo preklopi iz visokega na nizek logični nivo, vse podrejene enote začnejo meriti čas trajanja nizkega logičnega nivoja, ki je časovno točno določen glede na izbrano hitrost prenosa. Podrejene enote linijo držijo na nizkem logičnem nivoju dokler ta čas ne poteče. Šele ko vse podrejene enote zaključijo del periode na nizkem logičnem nivoju in sprostijo SCL linijo, lahko nadrejena enota postavi linijo nazaj na visok logični nivo (Slika 3.16). Del periode na nizkem logičnem nivoju tako določa podrejena enota, ki ima najdaljši čas dela periode na nizkem logičnem nivoju. Podrejene enote s krajšim časom nizkega logičnega nivoja so takrat v stanju čakanja na visok logični nivo.



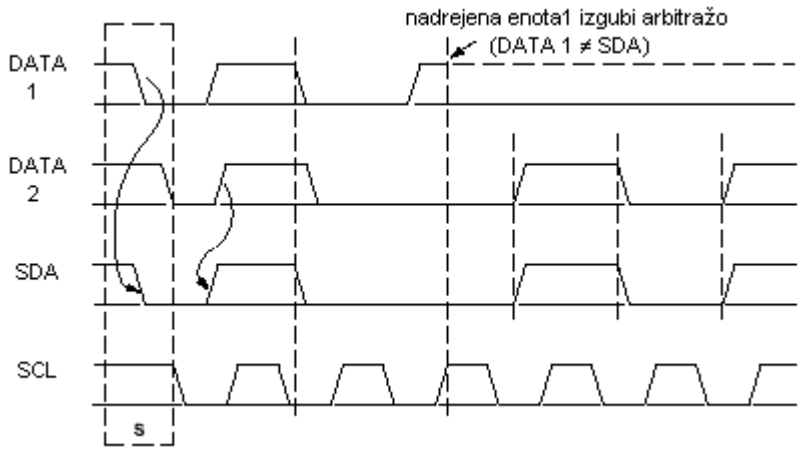
Slika 3.16: Sinhronizacija SCL signala med vsemi napravami na vodilu

Ko nadrejena enota preklopi na visok logični nivo, podrejene enote začnejo meriti del periode na visokem logičnem nivoju. Linija se postavi na nizek logični nivo, takoj ko ena podrejena enota odmeri del periode na visokem logičnem nivoju in linijo postavi na nizek logični nivo.

SCL linija je torej sinhronizirana tako, da je del periode na nizkem logičnem nivoju določen s podrejeno enoto z najdaljšim zahtevanim časom nizkega logičnega nivoja, del periode na visokem logičnem nivoju pa s podrejeno enoto z najkrajšim zahtevanim časom visokega logičnega nivoja.
Podrejene enote lahko to zmožnost, da SCL linijo zadržijo na nizkem logičnem nivoju, uporabijo za upočasnitev prenosa.
V visokohitrostnem režimu se prenos lahko zadrži le na bajtnem nivoju.

3.5.2.2 Arbitražna procedura

Nadrejena enota lahko začne nov prenos le, če je vodilo prosto. Zgodi se, da prenos začneta hkrati dve ali več nadrejeni enoti, tako da generirata START pogoja v nekem minimalnem časovnem intervalu, 'hold' času ($t_{HD,\ STA}$). Ostale naprave se odzovejo kot na običajen START pogoj. Arbitražna procedura poskrbi, da v tem primeru ostane aktivna le ena nadrejena enota, ostale nadrejene enote se odklopijo od vodila in počakajo, da je vodilo prosto.
SDA izhodi so na vodilo povezani z logično IN funkcijo. Če npr. nadrejena enota 1 pošlje na linijo nizek logični nivo, nadrejena enota 2 pa visok logični nivo, bo linija ostala na nizkem logičnem nivoju. Nadrejena enota 2 bo to zaznala in se odklopila iz linije oz. 'bo izgubila arbitražo'.
Najprej se primerjajo naslovni biti. Če obe nadrejeni enoti naslavljata isto podrejeno enoto, se arbitražna procedura nadaljuje pri podatkovnih bitih in potrditvah.
Nadrejena enota, ki je izgubila arbitražo, lahko nadaljuje z generiranjem SCL impulzov do konca bajta, v katerem je izgubila arbitražo.
Nadrejena enota v Hs režimu na začetku pošlje posebno 8-bitno rezervirano kodo, značilno le za eno nadrejeno enoto. Zato arbitražna procedura vedno učinkuje že pri prvem bajtu, takoj za START pogojem.
Nekatere nadrejene enote imajo možnost delovanja tudi kot podrejene enote. Če taka nadrejena enota med prenosom izgubi arbitražo, je možno, da ga prednostna nadrejena enota hoče nasloviti. Zato mora takoj po izgubi arbitraže preklopiti na režim podrejene enote.
Slika 3.17 kaže arbitražno proceduro za dve nadrejeni enoti. Nadrejena enota 1 generira signal DATA 1, nadrejena enota 2 generira signal DATA 2. V določenem trenutku nadrejena enota 1 generira visok logični nivo, nadrejena enota 2 pa nizek logični nivo. Takrat SDA linija ostane na nizkem logičnem nivoju, nadrejena enota 1 pa izgubi arbitražo.

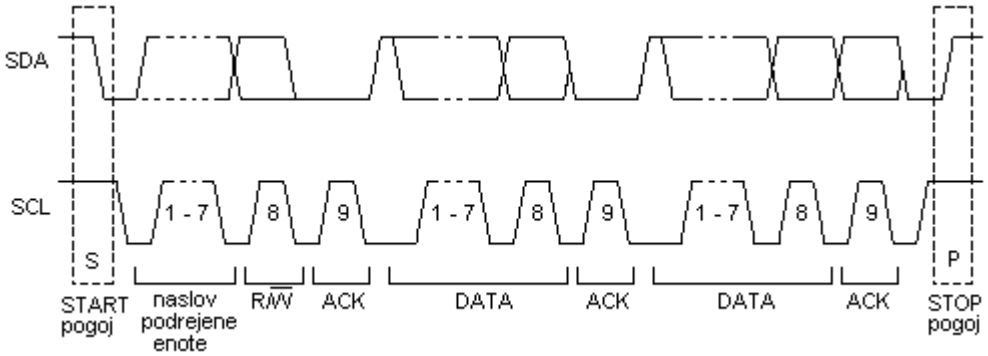


Slika 3.17: Arbitražna procedura

Katera nadrejena enota ima prednost je torej odvisno od trenutnega signala, zato pri I^2C vodilu ni vnaprej določenih prednostnih nadrejenih enot.
Poseben primer je, če se v F/S režimu arbitražna procedura odvija še med ponovljenim START pogojem ali STOP pogojem. Takrat arbitražna procedura ne učinkuje, kar moramo vnaprej upoštevati. V Hs režimu arbitražna procedura vedno učinkuje.

3.5.3 7-BITNO NASLAVLJANJE

Prenos se začne s START pogojem (S), sledi naslov podrejene enote, sestavljen iz sedmih bitov. Osmi bit (R/W) določi smer prenosa nadaljnih podatkov. Če je osmi bit '0', bo nadrejena enota pošiljala podatke podrejeni enoti (WRITE), če je '1', bo nadrejena enota brala podatke iz podrejene enote (READ). Prenos se vedno zaključi tako, da nadrejena enota pošlje STOP pogoj (P). Lahko pošlje tudi ponovljen START pogoj (Sr), če takoj nadaljuje s prenosom. Potek je prikazan na sliki 3.18.



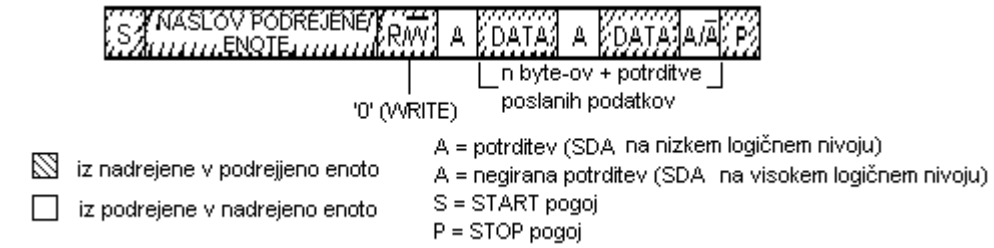
Slika 3.18: Prenos podatkov iz nadrejene enote v podrejeno enoto

Možni poteki prenosa podatkov:

- WRITE funkcija

Nadrejena enota samo pošilja podatke (R/W = 0). Nadrejena enota pošlje START pogoj (S), 7 – bitni naslov in '0' (WRITE). Podrejena enota pošlje potrditev. Potem nadrejena enota pošilja podatke in za vsak bajt podrejena enota generira potrditev. Ko hoče nadrejena enota zaključiti

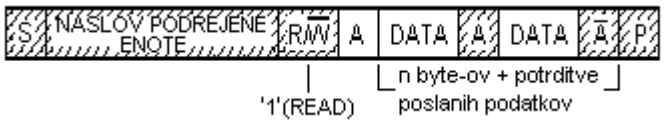
prenos, pošlje STOP pogoj oz. ponovljen START pogoj, če takoj nadaljuje s prenosom (Slika 3.19).



Slika 3.19: Nadrejena enota pošilja podatke podrejeni enoti s 7-bitnim naslovom in nespremenljivo smerjo pretoka podatkov

- READ funkcija

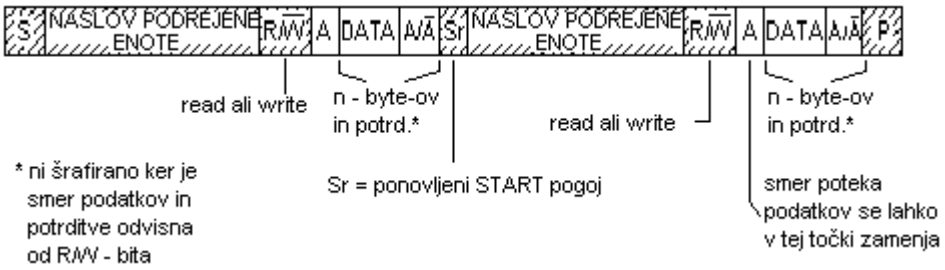
Nadrejena enota samo bere podatke (R/W = 1) . Nadrejena enota pošlje START pogoj, 7 – bitni naslov in '1' (READ). Podrejena enota generira potrditev in začne pošiljati podatke. Za vsak bajt nadrejena enota generira potrditev. Ko hoče nadrejena enota zaključiti prenos, pošlje negirano potrditev (impulz visokega logičnega nivoja) in STOP pogoj (Slika 3.20).



Slika 3.20: Nadrejena enota bere podatke iz podrejene enote

- Kombiniran potek READ / WRITE

Nadrejena enota bere in pošilja podatke. Pri spremembi smeri pretoka podatkov mora nadrejena enota ponoviti celoten prenos, tudi če gre za isto podrejeno enoto. Namesto STOP in takojšnjega START pogoja pošlje kar ponovljen START pogoj (Slika 3.21).



Slika 3.21: Nadrejena enota vpisuje in bere podatke iz podrejene enote

Primer uporabe kombiniranega poteka:

Kombiniran potek je uporabljen pri serijskih pomnilnih integriranih vezjih, npr. EEPROM-ih. Nadrejena enota - oddajnik pošlje lokacijo spominske celice (WRITE funkcija). Po spremembi smeri pomnilno integrirano vezje pošlje podatke nadrejeni enoti (READ funkcija).

Nekatera pomnilna integrirana vezja samodejno povečajo naslov lokacije spominskega prostora za nadaljne branje. To možnost določa proizvajalec spominskih integriranih vezjih.

Nekateri od sedmih bitov v naslovu so stalno določeni, nekatere pa lahko izbiramo. Izbirne bite določimo s priključnimi sponkami na integriranem vezju. S številom izbirnih bitov je določeno največje število možnih naslovov in s tem naprav na vodilu. Npr. če ima naprava tri izbirne priključne sponke, je na vodilu lahko največ osem istih naprav. Nespremenljive bite določijo proizvajalci posameznih vezij.

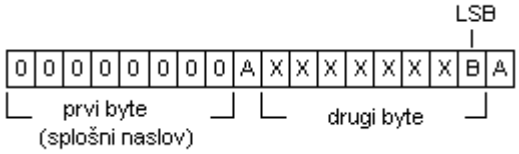
Nekaj naslovov je rezerviranih:

Tabela 3.2 : Rezervirani naslovi za posebna navodila podrejenim napravam

Naslov	**R/W - bit**	**Opis**
0000 000	0	Splošni naslov
0000 000	1	START bajt
0000 001	X	CBUS naslov
0000 010	X	Za druge vrste protokolov
0000 011	X	Za nadaljne razvojne namene
0000 1XX	X	Naslov nadrejene enote v Hs režimu
1111 1XX	X	Za nadaljne razvojne namene
1111 0XX	X	10 - bitno naslavljanje

3.5.3.1 Splošni naslov (angl. general call address)

Kadar hoče nadrejena enota poslati podakte vsem napravam hkrati, uporabi splošni naslov. Sestavljen je iz dveh bajtov. Prvi bajt je 0000 0000. V drugem bajtu so nadaljna navodila za vse podrejene enote (Slika 3.22).



Slika 3.22: Splošni naslov

Naprave lahko ta naslov tudi ignorirajo, tako da ne pošljejo potrditve. Naprave, ki se odzovejo, pošljejo potrditev in delujejo kot podrejene enote sprejemniki. Za naslednje bajte vse sodelujoče naprave pošiljajo potrditve. Če neka naprava podatkov ne sprejema več, nadrejena enota pa še kar pošilja bajte, naprava pošlje negirano potrditev.

Pri drugem bajtu ločimo dva primera, glede na LSB bit :

1. LSB = '0'
   - 0110 (H'06').

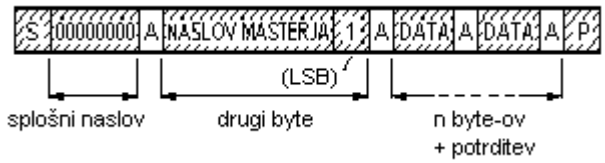
Podrejene enote se resetirajo in vpišejo izbirne naslovne bite. Pri resetu je treba paziti, da naprave ne povlečejo SCL ali SDA linije na nizek logični nivo, ker bi to ustavilo prenos.

   - 0100 (H'04').

podrejene enote vpišejo izbirne naslovne bite, brez reseta.

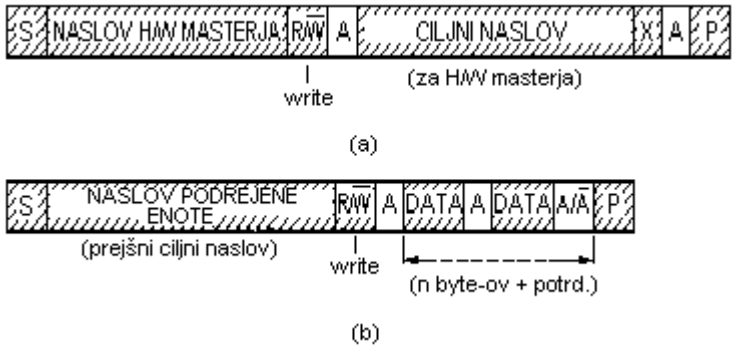
Kombinacija 0000 0000 (H'00') ni dovoljena za drugi bajt. Ostale kombinacije naprave ignorirajo.

2. LSB = '1'
Strojni splošni naslov. Nadrejena enota, ki generira ta naslov, ne določi, kateri podrejeni enoti so podatki namenjeni, ampak samo pove svoj naslov. Podatke bere katerakoli podrejena enota, ki te podatke potrebuje. Taka nadrejena enota se imenuje strojna nadrejena enota (angl. hardware ali H/W master). Primer strojne nadrejene enote je tipkovnica. Strojna nadrejena enota lahko deluje tudi kot podrejena enota, v tem primeru ima naslov, ki ga pošlje ob strojnem splošnem naslovu (Slika 3.23).



Slika 3.23: Splošni naslov strojne nadrejene enote

V nekaterih primerih strojna nadrejena enota ob resetu celotnega sistema najprej začne delovati kot podrejena enota. Neka druga nadrejena enota mu pošlje naslov naprave, kamor mora poslati podatke. Strojna nadrejena enota dalje deluje kot nadrejena enota - oddajnik (Slika 3.24).

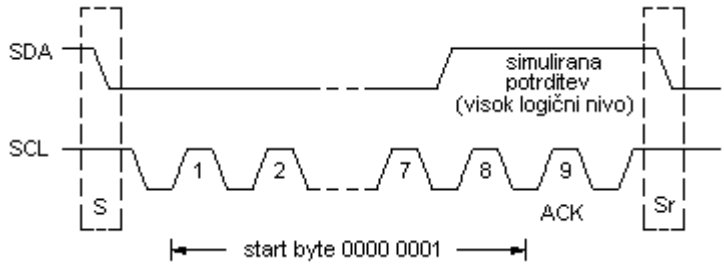


Slika 3.24: Strojni master (ob resetu v režimu sprejemnika)
(a) Strojni master dobi naslov, kamor mora poslati podatke
(b) Strojni master pošlje podatke

3.5.3.2 START bajt

Nekateri mikrokrmilniki imajo vgrajen I^2C priključek, kar obremenjuje delovanje programa le kadar je prisoten I^2C signal. Če je vodilo priključeno na navadne I/O priključke, jih mora programsko stalno spremljati, kar upočasnjuje njegovo delovanje. Delno ga lahko razbremenimo z uporabo start bajta (Slika 3.25). Celotna start procedura v takem primeru izgleda takole:

- START pogoj (S)
- START bajt (0000 0001)
- Potrditev
- Ponovljen START pogoj (Sr)

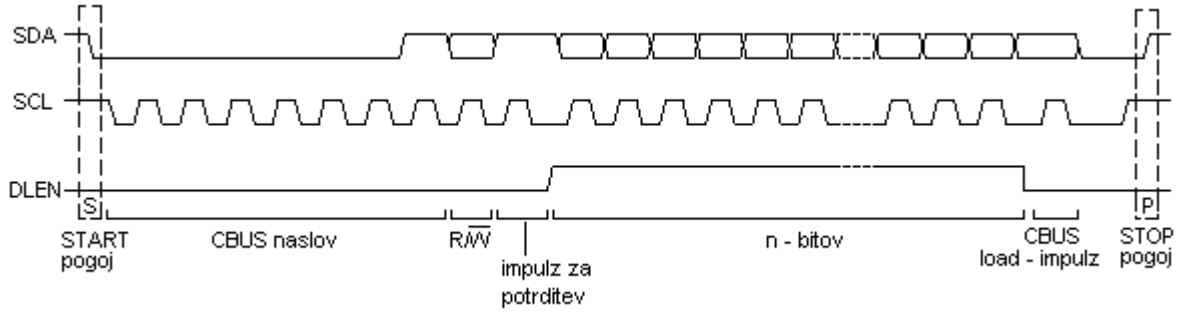


Slika 3.25: START bajt

Nadrejena enota najprej pošlje START pogoj in nato START bajte (0000 0001). Mikrokrmilnik lahko vzorči I/O sponke pri nižji vzorčni frekvenci, le da najde eno od sedmih ničel. Potem preklopi na hitrejše vzorčenje, da najde ponovljen START pogoj, s katerim sinhronizira nadaljni potek.
Po START bajtu nadrejena enota generira tudi SCL pulz za potrditev, vendar START bajta naprave ne potrjujejo.
V primeru strojne nadrejene enote v režimu sprejemnika se le-ta pri ponovljenem START pogoju resetira in tako ignorira START bajt.

3.5.3.3 Kompatibilnost s CBUS protokolom

CBUS protokol je podoben I^2C protokolu, prenos podatkov pa poteka po treh linijah. CBUS sprejemniki se lahko priključijo na I^2C vodilo na standardni hitrosti. Dodana mora biti še tretja linija DLEN, potrditev pa izpuščena. Za I^2C so običajno podatki v osem bitnem formatu, CBUS pa ima različne formate.
Če je vodilo grajeno za I^2C in CBUS protokol, se I^2C naprave ne smejo odzivati na CBUS signal. Zato je za CBUS rezerviran naslov (0000 001X). Po oddaji CBUS naslova, se vklopi DLEN linija, potem lahko steče CBUS protokol (Slika 3.26). STOP pogoj prepoznajo vse naprave, tudi I^2C, in ponovno čakajo na START pogoj.



Slika 3.26: Protokol pri CBUS oddajniku/sprejemniku

Pri dani stalni CBUS konfiguraciji se lahko čase zakasnitev prilagodi uporabljenim integriranim vezjem.

3.5.3.4 Ostale vrste protokolov

Naslov 0000 010X je rezerviran za druge vrste komunikacijskih protokolov, ki se lahko priključijo in izvajajo na I^2C vodilu. Na ta naslov se odzovejo le tiste I^2C naprave, ki so grajene tudi za te posamezne vrste protokolov.

3.6 CAN

CAN (Controller Area Network) je standard za serijski prenos podatkov, ki se uporablja v sistemih, ki delujejo v **realnem času**. Protokol je razvila firma Robert Bosch GmbH za komunikacijo med različnimi elektronskimi komponentami v avtomobilu in je alternativa drugim zahtevnejšim načinom ožičenja. Zaradi robustnosti se ta protokol uporablja tudi v drugih sistemih industrijske avtomatizacije.

Glavne značilnosti CAN protokola so:
- izvaja se v realnem času,
- hitrost prenosa je do 1 Mbit/s,
- 11-bitno naslavljanje,
- možnost detekcije napake.

CAN prenos je dokumentiran v standardu ISO 11898 (za hitre aplikacije) in v ISO 11519-2 (za nižjo hitrost prenosa podatkov).
CAN komunikacijo najpogosteje zasledimo v avtomobilski tehniki, v dvigalih, v kopirnih strojih, v krmilnikih v proizvodnji in v medicinski tehniki.

CAN komunikacija poteka po enem paru prepletenih žic. Pri CAN ni natančno specificirana postavitev in struktura vodila. Namesto tega se zahteva, da je naprava, ki je priključena na vodilo, sposobna oddajati oz. sprejemati enega od dveh signalov – nadrejenega ali podrejenega. Nadrejeni signal je lahko na eni podatkovni liniji predstavljen kot logična nič, podrejeni pa kot logična ena.

V primeru, da ena naprava vsiljuje nadrejeni signal, druga naprava pa podrejenega, mora biti zagotovljeno, da prevlada nadrejeni signal. Če to velja, protokol določa format podatkovnega paketa in prenosna pravila glede prioritete pošiljanja sporočil, zagotavlja zakasnitvene čase, obravnava napake pri prenosu in ponovno pošiljanje »pokvarjenih« sporočil. Razlikuje tudi med trajno in začasno napako v vozlišču. CAN protokol omogoča tudi večje število »master« enot v sistemu.

3.7 Fire Wire

To je zmogljivo serijsko vodilo, ki ga je razvilo podjetje Apple Computers Inc. Ta način serijske komunikacije obravnava standard IEEE 1394. Protokol je odgovor na vedno večjo potrebo po masovnem prenosu podatkov. Lokalna omrežja – LAN in globalnejša omrežja (Wide Area Network) ne omogočajo cenene povezave in zadostne pasovne širine za aplikacije, ki delujejo v realnem času.
Značilnosti Fire Wire protokola so:
- hitrosti prenosa so od 12,5 do 400 Mbit/s,
- omogočajo 64-bitno naslavljanje,
- vključevanje in izključevanje v realnem času (plug and play),
- paketno strukturiran prenos .

I^2C in CAN sta protokola za povezovanje integriranih vezij, Fire Wire pa je namenjen povezovanju neodvisnih elektronskih naprav (npr. osebni računalnik – scanner). Lahko podpira celotno lokalno omrežje, podobno kot Ethernet.

64 bitni naslov je razdeljen na:

- 10 bitov za identifikacijo omrežja,
- 6 bitov za identifikacijo vozlišča in
- 48 bitov za naslove spomina.

Lokalno omrežje, ki je zasnovano na Fire Wire protokolu, lahko obsega 1023 podomrežij, od katerih lahko vsako sestavlja 63 vozlišč, pri tem pa vsako vozlišče lahko naslavlja 281 Tbajtov lokacij.
Fire Wire komunikacija je uporabna za pogone diskov, za tiskalnike, scannerje, kamere in drugo širokopotrošno elekroniko.

3.8 USB

USB vmesnik omogoča uporabniku enostavnejše priključevanje monitorjev, tiskalnikov, modemov, digitalnih kamer ipd. Obstajata dve hitrosti prenosa podatkov:
12 Mbit/s za naprave s širšo pasovno širino in
1,5 Mbit/s za naprave z ožjo pasovno širino.

USB ima topologijo v obliki zvezde, kar pomeni, da vozlišča (hubi) lahko služijo kot povezovalne enote z USB perifernimi enotami. Le ena naprava mora biti priključena na osebni računalnik. Ostale naprave so priključene na vozlišče. Na ta način je lahko priključenih do 127 naprav.
Pri priključevanju naprav se vse izvede avtomatsko (nastavitev pasovne širine, gonilniki,..) Poleg izmenjave informacij se periferne enote lahko preko vodila tudi napajajo (do 5 m).

4. Paralelni prenos podatkov

4.1 IEEE – 488 vmesnik

To paralelno vodilo je bilo prvotno razvito za povezavo in krmiljenje instrumentov s pomočjo mikroračunalnika. Pozneje pa se je izkazal kot zelo zanesljiv način povezovanja tudi drugih naprav procesne tehnike v obsežnejše sisteme.
To vodilo je prvotno razvilo podjetje Hewlett-Packard za kominikacijo njihovih instrumentov z nadzornim računalnikom. Vmesnik so poimenovali Hewlett-Packard Interface Bus – HPIB. Ta način prenosa so z manjšimi popravki standardizirali leta 1975 pod imenom Digital Interface for Programmable Instrumentation IEEE - 488. Popravljena izdaja tega standarda je izšla leta 1978. Lahko rečemo, da je IEEE - 488 danes eden popolnejših standardov tako glede mehanskih, električnih in funkcionalnih specifikacij, kot tudi glede same oblike oddajanja in sprejemanja podatkov. Poleg imena IEEE - 488 se pojavlja še nekaj imen, kot npr. IEC vodilo, HP-IB vodilo, GPIB (General Purpose Interface Bus).

4.1.1 Pomembnejša določila in omejitve standarda IEEE - 488

Standard IEEE - 488 je zelo obsežen. Glavne omejitve, ki jih standard predpisuje, so naslednje:

- prenos podatkov med napravami je digitalen,
- na eno IEEE – 488 vodilo lahko priključimo največ 15 naprav,
- kabli za prenos so lahko dolgi največ 20 m,
- hitrost prenosa po katerikoli liniji ne sme presegati 1 Mbit/s.

Važnejša določila standarda pa so naslednja:

- definira splošno uporabni sistem z omejeno razdaljo med perifernimi enotami,
- definira od naprave neodvisne mehanske, električne in funkcionalne zahteve za vmesnik (kabli, konektorji, logični nivoji, tokovne in napetostne zahteve, funkcije vmesnika, handshake protokol itd.),
- specificira terminologijo in definicije IEEE - 488 vodila.

4.1.2 Prednosti uporabe IEEE – 488 vodila pri povezovanju naprav v sistem

Prednosti sistema, ki uporablja IEEE – 488 vodilo, pred klasičnimi ročnimi in avtomatskimi načini procesiranja so naslednje:

- bolj zanesljivi rezultati pri ponavljajočem zbiranju informacij, pri čemer rezultati niso odvisni od operaterjevih napak,
- večja hitrost meritev oz. prenosa podatkov,
- kompleksnejše testiranje oz. merjenje, ker lahko zaradi velike hitrosti merimo bistveno več parametrov v krajšem času,
- sistematičen prikaz rezultatov (npr. v določenih sistemskih enotah),
- večja natančnost zaradi možnosti sistematičnega upoštevanja napak v rezultatih,

- večja prilagodljivost (pri določenih nenormalnostih je možno uporabiti druge algoritme),
- možnost povezovanja naprav različnih proizvajalcev v enoten sistem,
- možnost neposredne komunikacije med enotami brez pretvornikov ali dodatnih kontrolni podsistemov.

4.1.3 Vrste naprav, ki jih priključujemo na IEEE – 488 vodilo

Naprave, ki jih priključujemo na IEEE – 488 vodilo, imajo lahko naslednje lastnosti:

Sprejemanje (LISTENER)
Enota lahko samo sprejema informacije od drugih na IEEE – 488 vodilo priključenih enot. (npr. programirljivi usmernik).

Oddajanje (TALKER)
Enote lahko samo oddajajo informacije drugim enotam na IEC vodilu (npr. digitalni voltmeter).

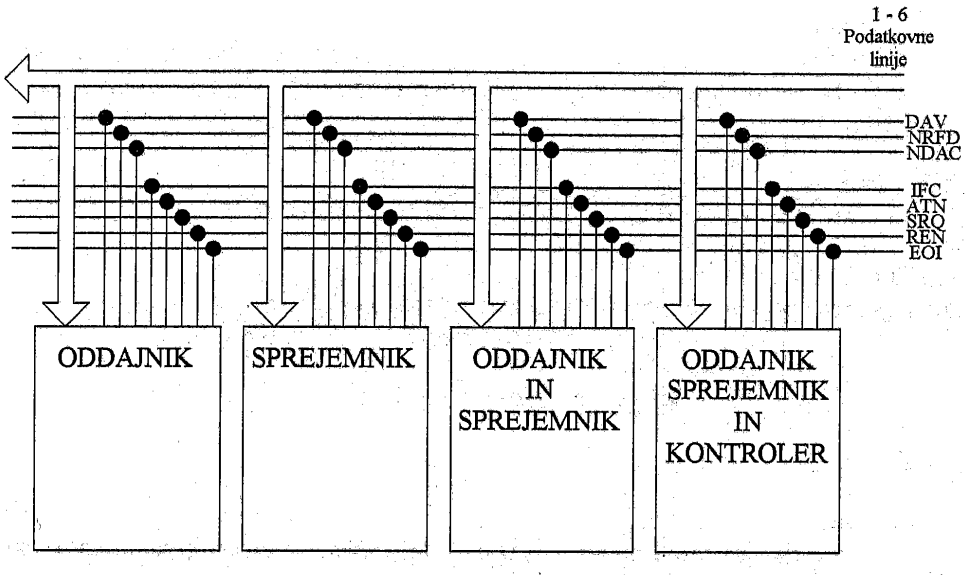
Sprejemanje in oddajanje (TALKER AND LISTENER)
Enota lahko sprejema ali oddaja informacije na IEC vodilo (npr. programirljivi digitalni voltmeter, ki mu lahko preko vodila določimo način delovanja).

Krmiljenje (CONTROLLER)
Enota naslavlja druge enote na IEC vodilu za oddajo ali sprejem in določa načine delovanja teh enot (mikroračunalnik, mikrokrmilnik, osebni računalnik).

Istočasno je lahko na celotnem vodilu aktiven le en oddajnik. Minimalna konfiguracija sistema vsebuje en oddajnik in en sprejemnik. Vsak kompleksnejši merilno-procesni sistem pa mora seveda vsebovati nadzorno enoto (računalnik).

Vsaka naprava na vodilu ima svoj naslov, ki ga običajno nastavimo s stikali na zadnji strani naprave. S stikali lahko izberemo tudi ti. »TALK only« in »LISTEN only« funkciji (delovanje v lokalnem načinu). Ti dve funkciji se običajno uporabljata, če v sistemu ni nadzorne enote. Pri tem načinu lahko neposredno povežemo oddajnik v TALKonly delovanju z enim ali več sprejemniki v LISTEN only delovanju (npr. A/D pretvornik in tiskalnik).

Na sliki 4.1 je prikazan sistem naprav in strukturo IEEE – 488 vodila.



Slika 4.1: Struktura IEEE-488 vodila

4.1.4 Struktura IEEE – 488 vodila

IEEE – 488 vodilo je sestavljeno iz 16 signalnih linij, ki jih lahko razdelimo v tri vodila:

- podatkovno vodilo (8 linij),
- nadzorno vodilo (3 linije),
- krmilno vodilo (5 linij).

Preko podatkovnega vodila se prenašajo 7 bitna sporočila iz vmesnika (ASCII kode) in 8 bitni merilni podatki. Podatki se prenašajo bitno paralelno, bajtno serijsko, komunikacija je asinhrona, prenos podatkov pa je dvosmeren.

Po podatkovnih linijah se prenašajo:

- naslovi,
- ukazi enotam na IEC vodilu,
- merilni podatki,
- splošni ukazi in podatki stanj.

Nadzorno vodilo nadzoruje prenos podatkov po podatkovnem vodilu in omogoča tako imenovano handshake komunikacijo med posameznimi vmesniki oz. napravami. Na ta način je možna asinhrona komunikacija naprav z različno hitrostjo. Linije so naslednje:

DAV (data valid) – označuje veljavnost podatka, ki ga oddaja oddajna naprava.

NRFD (not ready for data) – označuje nepripravljenost naprave za sprejemanje podatkov,

NDAC (not data accepted) – označuje, da podatek v sprejemni napravi še ni sprejet.

Krmilno vodilo pa vsebuje naslednje linije:

ATN (Attention) – ukaz prihaja od nadzorne enote in pove, kako naj se interpretirajo podatki na podatkovnih linijah,

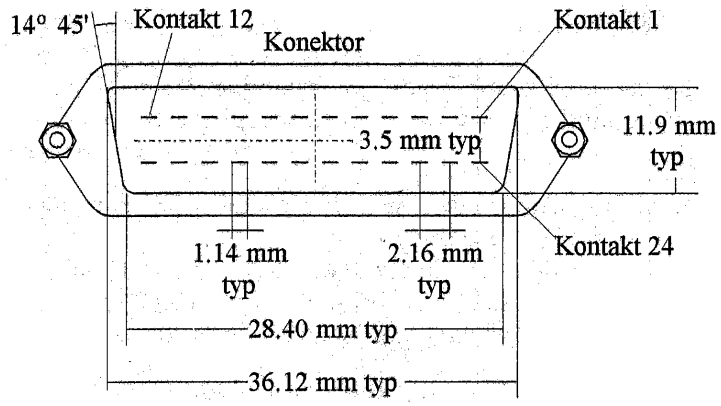
IFC (Interface Clear) – ukaz prihaja od nadzorne enote in postavi vmesnike vseh naprav v začetno stanje,

SRQ (Service Request) – to sporočilo odda naprava v sistemu, ko potrebuje določeno akcijo nadzorne enote (npr. če naprava ne razume sporočila, če tiskalniku zmanjka papirja ipd.),

EOI (End of Identify) – linija označuje konec večbajtnega prenosa podatkov, v zvezi z ATN pa omogoča parallel poll funkcijo,

REN (Remote Enable) – omogoča delovanje vmesnika v načinu LOCAL (npr. napravo ročno programiramo) ali REMOTE (napravo programiramo preko IEC vodila),

Slika 4.2 kaže obliko konektorja, preko katerega se povezujejo naprave in razporeditev linij na konektorju.



Kontakt	Linija	Kontakt	Linija
1	DIO1	13	DIO5
2	DIO2	14	DIO6
3	DIO3	15	DIO7
4	DIO4	16	DIO8
5	EOI	17	REN
6	DAV	18	Gnd, (6)
7	NRFD	19	Gnd, (7)
8	NDAC	20	Gnd, (8)
9	IFC	21	Gnd, (9)
10	SRQ	22	Gnd, (10)
11	ATN	23	Gnd, (11)
12	OKLOP	24	Gnd, LOGIC

Slika 4.2: Priključki IEEE-488 vodila

4.1.5 Način delovanja vmesnika

Vmesnik deluje po točno določenem zaporedju, ki je definirano s standardom IEEE – 488. Delovanje vmesnika je določeno s skupino med seboj povezanih stanj ter z logičnimi izrazi, ki so potrebni za prehod med stanji. Najbolj popolna oblika vmesnika ima 10 različnih načinov delovanja, vsak način delovanja pa je opisan z diagramom prehajanja stanj. Posamezni načini delovanja so naslednji:

Source Handshake (SH)
– krmili začetek in konec oddaje večlinijskega sporočila,

Acceptor Handshake (AH)
– omogoča pravilen sprejem oddaljenega večlinijskega sporočila,

Talker (T), Talker Extended (TE)
– omogoči napravi prenos podatkov drugim napravam na IEEE – 488 vodilu,

Listener (L), Listener Extended (LE)
– omogoča napravi sprejemanje podatkov z IEEE – 488 vodila,

Service Request (SRQ)
– omogoča napravi zahtevo po »servisu«,

Remote Local (RL)
– omogoča napravi izbiranje ned dvema viroma informacij: vir je lahko lokalni (npr. informacije, ki jih dobi naprava iz sprednje, kontrolne plošče) ali pa oddaljeni (informacije oz. programske kode, ki jih naprava dobi po IEEE – 488 vodilu),

Parallel Poll (PP)
– omogoča napravi, da pošlje en statusni bit nadzorni napravi, ne da bi bila pred tem naslovljena kot oddajna naprava,

Device Clear (DC)
– omogoča inicializacijo naprave,

Device Trigger (DT)
– omogoča enoti izvršitev določene akcije (npr. A/D pretvornik vzorči napetost),

Controller (C)
– omogoča pošiljanje naslovov, univerzalnih ukazov, naslovnih ukazov na IEEE – 488 vodilo. To funkcijo ima v sistemu računalnik.

4.1.6 Naslavljanje in prenos podatkov na IEEE – 488 vodilu

Bistvena funkcija merilno procesnega sistema je seveda prenos podatkov od oddajnika proti sprejemnikom. Toda nadzorna enota (računalnik) mora že pred tem poskrbeti za vse porebno, da bodo podatki res tisto, kar si želi uporabnik sistema. Bistvena funkcija nadzorne enote je naslavljanje naprav v sistemu. Vsaka naprava ima v sistemu svoj naslov (0 – 30), ki se nastavi

na sami napravi. Nadzorna enota na določene naslove pošilja ukaze in na ta način pripravi sistem za pravilno delovanje.

Vendar pri naslavljanju nimamo nikakršnega posebnega naslovnega vodila, ampak imamo na voljo le 8 bitno IEEE – 488 vodilo. Da ločimo med naslovno informacijo in podatki, uporablja nadzorna enota ATN (Attention) linijo. Če je ATN = 0, potem 8 bitni podatek na vodilu predstavlja en znak (običajno ASCII) podatka. Kadar pa je ATN = 1, pomeni 8 bitna informacija na vodilu ukaz ali pa naslov, ki ga pošilja nadzorna enota. V tem primeru imajo pomen le biti od 0 do 6. Bita 5 in 6 pa povesta, ali gre za naslov ali ukaz. Velja naslednja tabela:

biti	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
ukaz	X	0	0	ukaz				
LISTEN naslov	X	0	1	naslov sprejemnika				
TALK naslov	X	1	0	naslov oddajnika				

Oglejmo si enostaven primer programiranja digitalnega multimetra HP 3490A. IEEE – 488 standard seveda ne predpisuje, kakšne znake (ASCII) je treba poslati napravi, da jo ustrezno programiramo. To mora poiskati uporabnik v priročniku ustrezne naprave. Če pošiljamo digitalnemu multimetru 3490A ASCII znake F2R3, programiramo instrument v funkcijo 2 (AC volti) in področje 3 (range 100 V). Tako proceduro lahko običajno naredimo v višjih proramskih jezikih. Npr. v BASIC-u naredimo to z eno vrstico programa:

OUTPUT 723, »F2R3«
Pri tem je 7 naslov IEEE – 488 vodila (pomembno, če je povezanih več vodil oz. več nadzornih enot – računalnikov), 23 pa je naslov naprave 3490A. Tudi računalnik kot nadzorna enota, sprejemnik in oddajnik hkrati ima svoj naslov...21.

Izvršitev stavka v jeziku BASIC bo povzročila naslednjo sekvenco sporočil po 8 bitnem vodilu:

ATN	**Podatkovne linije**	**ASCII**	**Pomen**
1	00111111	?	UNLISTEN ukaz (onemogoči vse prejšnje sprejemnike)
1	01010101	U	računalnik (21) je oddajnik
1	00110111	7	naprava 23 je sprejemnik
0	01000110	F	prvi podatkovni bajt
0	00110010	2	drugi podatkovni bajt
0	01010010	R	tretji podatkovni bajt
0	00110011	3	četrti podatkovni bajt
0	00001101	CR	carriage return
0	00001010	LF	line feed

Računalnik deluje v prvem delu kot nadzorna enota. Najprej pošlje UNLISTEN ukaz, s katerim onemogoči vse prejšnje sprejemnike. Nato pošilja naslove pri stanju na liniji ATN = 1. Najprej pošlje sebi svoj TALK naslov (21), digitalnemu multimetru pa njegov LISTEN naslov (23). Vse to računalnik torej opravi tako, da ob liniji ATN = 1 pošlje tri ASCII znake ?U7. Nato računalnik (nadzorna enota) postavi ATN linijo v ATN = 0. Ob prehodu preide računalnik v aktivno TALK stanje, digitalni multimeter pa v aktivno LISTEN

stanje. S tem je vse pripravljeno, da računalnik pošlje napravi programske kode, to so v našem primeru štiri ASCII znaki F2R3. Večbajtno sporočilo se vedno zaključi na nek predpisan način, v našem primeru z dvema znakoma CR in LF. Večbajtno sporočilo je možno zaključiti tudi s posebno linijo krmilnega vodila EOI, ki jo postavimo v EOI = 1 pred prenosom zadnjega bajta. Na kakšen način zaključimo večbajtno sporočilo, je opisano v priročniku uporabljene naprave (včasih sta dani obe možnosti).

Čitanje napetosti iz digitalnega multimetra poteka z BASIC ukazom:

ENTER 723, A
Ob tem ukazu nadzorna enota spremeni multimeter iz sprejemnika v oddajnik, sebe pa iz nadzorne enote v sprejemnik. Digitalni multimeter pošilja merjeno napetost v obliki 15 ASCII znakov:

±D.DDDDDE±DD CR LF
Ustrezna numerična vrednost, ki jo predstavljajo ASCII znaki, se shrani v spremenljivko A.
Če ima programski jezik že vgrajeno operacijo za vodilo IEEE – 488, je programiranje zelo enostavno, v nasprotnem primeru pa si moramo sami izdelati ustrezne podprograme (najbolje v zbirniku), ki jih nato kličemo iz višjega programskega jezika.

4.2 PCI

PCI (Peripheral Component Interconnect) je namenjen povezovanju integriranih vezij, matičnih plošč, spominskih enot ipd. Protokol je v začetku devetdesetih let razvil Intel (v osebnih računalnikih leta 1994, Intel 486).
Glavne značilnosti so:

- hitrost prenosa je od 127,2 Mbit/s do 508,6 Mbit/s
- 32 bitno naslavljanje,
- vodilo je sinhrono,
- 32 mutipleksiranih podatkovno-naslovnih linij.