Univerzitet u Sarajevu
Elektrotehnicki fakultet Sarajevo

Bajramovic Kemal
Index br.12259

Zadatak 1.Pretvoriti heksadecimalni broj FACE u broj sa bazom 2,3,4 i 10.

Heksadecimalni broj se prevodi u binarni tako što svakoj cifri heksadecimalnog broja dodijelimo 4-bitni binarni kod,tako je,

$FACE_{16} = 1111\ 1010\ 1100\ 1110_2$.

Prevodjenje u decimalni oblik podrazumjeva razvoj broja na slijedeci nacin,

$FACE_{16} = 15*16^3 + 10*16^2 + 12*16^1 + 14*16^0 = 61440 + 2560 + 192 + 14 = 64206_{10}$.

Iz ovog oblika prevesti cemo broj u brojni sistem sa bazom 3 dijeljenjem prvo broja a zatim medjurezultata sa 3 i pamcenjem ostataka koje zatim citamo odozdo prema gore na slijedeci nacin,

$FACE_{16} = 64206_{10}$	:	3	=	21402	+0	
21402	:	3	=	7134	+0	
7134	:	3	=	2378	+0	
2378	:	3	=	792	+2	
792	:	3	=	264	+0	
264	:	3	=	88	+0	
88	:	3	=	29	+1	
29	:	3	=	9	+2	
9	:	3	=	3	+0	
3	:	3	=	1	+0	
1	:	3	=	0	+1	te je,

$FACE_{16} = 10021002000_3$.

Na isti nacin broj prevodimo u brojni sistem sa bazom 4 dijeljenjem sa 4 prvo broja a zatim medjurezultata,pamcenjem ostataka dijeljenja i njihovim citanjem odozdo prema gore,

$FACE_{16} = 64206_{10}$	:	4	=	16051	+2	
16051	:	4	=	4012	+3	
4012	:	4	=	1003	+0	
1003	:	4	=	250	+3	
250	:	4	=	62	+2	
62	:	4	=	15	+2	
15	:	4	=	3	+3	
3	:	4	=	0	+3	te je,

$FACE_{16} = 33223032_4$.

Zadatak 2. Prikazati sve korake pri sabiranju dvaju brojeva x i y u formatu pokretnog zareza jednostruke preciznosti. Koristiti binarne operacije i prikazati sve medjurezultate.Neka su, na pocetku, operandi u IEEE formatu. Voditi racuna o normalizaciji i zaokrudjivanju rezultata.

Floating point broj sastoji se od *sign* bita koji odredjuje da li je broj pozitivan (0) ili negativan (1) ,eksponenta i frakcije.Najznacajniji bit je *sign* bit, zatim slijedi eksponent pa frakcija što modjete vidjeti na donjoj slici,

Slika 2-1. Floating point broj

Postoji više formata floating point brojeva, a format jednostruke preciznosti podrazumjeva 1 bit za znak,8 bita za eksponent i 23 bita za frakciju,ukupno 32 bita.

Floating point broj u formatu jednostruke preciznosti se tumaci kao,

$$X = \text{frakcija} * 2^{(\text{eksponent}-127)}$$

Vidimo da je eksponent prikazan u Excess127 kodu.Ovo nam obezbjedjuje uvijek pozitivan eksponent s obzirom na operacije sabiranja i oduzimanja koje vršimo nad tim eksponentom kada izvodimo aritmeticke operacije sa floating point brojevima.Frakcija je normalizovana ako je eksponent podešen tako da je vodeci bit frakcije 1.Ovaj bit i binarna tacka se podrazumjevaju pa se u broju ne zapisuju,na taj nacin dobijamo bit na racun preciznosti.

Saberimo brojeve $X = 1{,}6875 * 2^{-55}$ i $Y = 1{,}25 * 2^{-57}$. Brojevi se u eksponentu ne razlikuju za mnogo da bi proces sabiranja trajao krace.

frakcija(X) = 1,6875 = 3,375/2 =6,75/4= 13,5/8 = 27/16 te ce u binarnom obliku biti

frakcija(X) = 10110.....0 (1-ca i binarna tacka se podrazumjevaju)

eksponent(X) = -55 + 127 = 72_{10} te ce u binarnom obliku biti

eksponent (Y) = 01001000 , pa je broj X zapisan u floating point formatu 1-preciznosti,

X = 0 01001000 10110000000000000000000 (*po dva blanck mjesta izmedju sign bita , eksponenta i frakcije radi lakšeg citanja)

frakcija (Y) = 1,25 = 2,5/2 =5/4 te ce u binarnom obliku biti ,

frakcija (Y) = 010.....0 (1-ca i binarna tacka se podrazumjevaju)

eksponent (Y) = -57 + 127 = 70_{10} te ce u binarnom obliku biti

eksponent (Y) = 01000110 te je broj Y zapisan u floating poin formatu 1-preciznosti,

Y = 0 01000110 01000000000000000000000

Prvi zadatak koji se namece je "poravnavanje" eksponenata ,tj.eksponenti moraju imati iste vrijednosti da bi se brojevi mogli sabrati (što je sasvim uobicajeno,prisjetimo li se kako sabiramo decimalne brojeve predstavljenje u $10^{\text{eksponent}}$ notaciji).

Ovo poravnavanje se modje izvesti zahvaljujuci slijedecem: Ako eksponentu dodamo broj 1 to ce biti kao da smo broj pomnodjili sa 2 ($2^{-57+1} = 2^{-57}*2 = 2^{-56}$), a ako frakciju šiftamo jedno mijesto udesno to je kao da smo broj podijelili sa 2 (opšte poznato za binarne brojeve).Dakle, ako eksponentu dodamo 1 a frakciju šiftamo za

jedno mjesto udesno to ce biti kao da smo broj pomnodjili sa 2 a zatim podijelili sa 2,dakle broj ce ostati isti.Ako ovo dva puta izvedemo nad brojem Y dobicemo broj Y izradjen preko eksponenta -55 što i tradjimo.

Y = 0 01000110 01000000000000000000000 ,dodajmo 1 eksponentu,šiftajmo frakciju,
+1 rshift(frakcija)
Y = 0 01000111 10100000000000000000000 ,dobili smo $Y = 0{,}625*2^{-56}$,ponovimo isto,
+1 rshift(frakcija)
Y = 0 01001000 01010000000000000000000 ,dobili smo $Y = 0{,}3125*2^{-55}$.

Dakle od broja $Y = 1{,}25*2^{-57}$ dobili smo $Y = 0{,}3125*2^{-55}$, što nam omogucuje regularno sabiranje sa brojem $1{,}6875*2^{-55}$. Saberimo ta dva broja binarno,(naravno sabiramo SAMO frakcije a eksponent ostaje isti - prepisujemo ga),

0 01001000 10110000000000000000000
+ 0 01001000 01010000000000000000000
0 01001001 00000000000000000000000

Zanimljivo,desio se prenos u eksponent pa smo dobili broj $Z = 1*2^{-54}$.
S druge strane je,

$1{,}6875 * 2^{-55}$
$+ 0{,}3125 * 2^{-55}$
$2{,}0000 * 2^{-55}$ a to je $1*2^{-55+1} = 1*2^{-54} = Z$. Dakle dobili smo tacan rezultat.

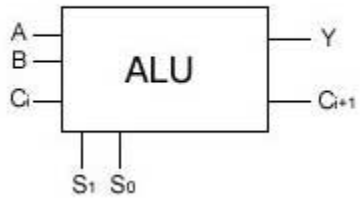
Medjutim šta je sa zaokrudjivanjem i normalizacijom što je spomenuto u postavci zadatka? U mom primjeru ocito nisam imao potrebe da vršim bilo zaokrudjivanje bilo normalizaciju.

Pretpostavimo da imamo potrebu frakciju šiftati toliko mijesta udesno (a modje i ulijevo ali bi tada od eksponenta trebali oduzimati 1 a ne dodavati 1) tako da na najmanje znacajnom bitu frakcije (23-cem) dodje 1. Ako šiftamo još jedno mijesto udesno izgubicemo na preciznosti. Medjutim šiftanje je nudjno , 1-cu cemo izgubiti a frakciju modjemo *zaokrudjiti* a to modjemo uraditi ostavljanjem 0 ili 1 na najmanje znacajnom bitu.Ako ostavljamo 0 mi smo uradili fakticki "otkidanje" a ako ostavimo 1 to znaci da smo frakciji dodali 1 (sabiranjem).
(Isti slucaj je i kod decimalnih brojeva npr.rezultat je 2,435535a mi modjemo prikazati samo 3 cifre u frakciji, tada je rezultat ili 2,435(otkidanje) 2,436(dodavanje 1)).

Normalizacija je potrebna jer se jedan broj u floating point formatu modje prikazati na više razlicitih nacina npr. $123 = 12{,}3 * 10^1 = 1{,}23 * 10^2$,što isto vrijedi i za binarne floating point brojeve.Da ne bi bilo nedoumica u tumacenju binarnih fp brojeva, kao normalizovan uzima se onaj broj kod kojeg frakcija pocinje sa podrazumjevanom jedinicom.
Šta to znaci? Pretpostavimo da smo sabirali dva broja X i Y kod kojih su eksponenti -2 i -1 respektivno.Algoritam je takav da oba broja moramo dovesti u formu eksponenta 0.Šiftanjem frakcije udesno odmah dobijamo nenormalizovane brojeve.I njihov zbir modje biti takodje nenormalizovan.Tada je potrebno uraditi slijedece,frakciju rezultata cemo šiftati ulijevo dok na najznacajnijem bitu nebude 1 (dakle broj je 0.1.....) tada cemo broj šiftati za još jedno mjesto (da bi dobili podrazumjevanu jedinicu) a od eksponenta cemo oduzeti broj jednak broju pomjeraja udesno prilikom šiftanja.Tako cemo dobiti rezultat u normalnoj formi.

Zadatak 3. Projektovati 1-bitnu ALU sa slijedecim operacijama nad ulazima A i B ,AIB, AVB, A+B, AxorB, korištenjem NILI kola.Strukturu nacrtati i simulirati u Electronic Workbenchu.Ako kola imaju kašnjenja 5ns koliko je maksimalno kašnjenje rezultata?

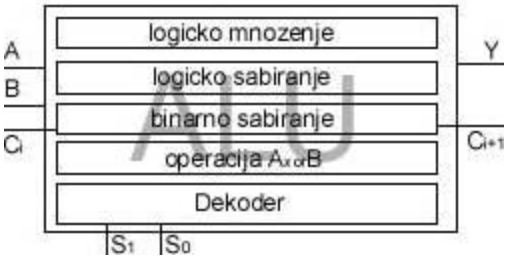
Ova ALU ocito mora imati 5 ulaza (dva za podatke A i B, dva kontrolna ulaza S_0 i S_1 koja odredjuju koja od 4 nabrojane operacije ce se izvesti nad tim podacima,jedan ulaz C_i za prenos u operaciji binarnog sabiranja A i B) te dva izlaza (Y predstavlja rezultat operacije nad A i B ,a C_{i+1} predstavlja eventualni prenos nastao u operaciji binarnog sabiranja A i B).
Napominjem da su prenosi nudjni ako,eventualno,djelimo vezivanjem *n* komada ALU ostvariti *n-bitnu* ALU.



Slika 3-1. ALU sa ulazima i izlazima

U principu postoje dva pristupa razvoju ALU.Prvi (mogli bismo ga nazvati *bottom-up* pristup) predvidja klasican nacin projektovanja jednog logickog sklopa.Imamo 5 ulaza ,dakle formirali bi tabelu istine koja bi imala $2^5 = 32$ vrste za sve moguce kombinacije ulaza.Zatim bi formirali dvije Karnoove mape za svaki od izlaza,dobili minimizirane funkcije koje bismo dodatno prilagodili upotrebi NILI kola i na koncu realizovali logicki sklop.
Drugi pristup (mogli bi ga nazvati *top-down* pristup) razvoju ove ALU zasniva se na tome da ALU posmatramo kao sklop sacinjen od svojih podsklopova ,svaki zadudjen da obavi svoju jedinstvenu funkciju.Iz razloga jednostavnosti i prakticnosti upravo cu upotrijebiti ovaj pristup razvoju ALU.

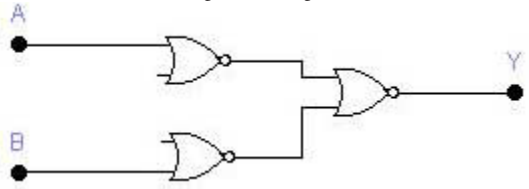
Ova ALU mora biti sacinjena od 5 podsklopova ,po jedan za svaku od cetiri operacije koje treba izvoditi, i jednog dekodera koji ce na osnovu signala S_0 i S_1 odrediti koja ce se operacija izvesti nad ulazima.



Slika 3-2. Dijelovi ALU

Logicko mnodjenje

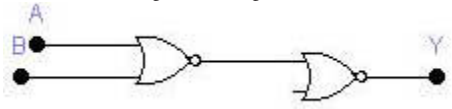
Ocito je AiB = (A' + B')' što se NILI kolima realizuje na slijedeci nacin,



Slika 3-3. AiB realizovano NILI kolima

Logicko sabiranje

Ocito je A+B = (A+B)" što se NILI kolima realizuje na slijedeci nacin,



Slika 3-4. AVB realizovano NILI kolima

Binarno sabiranje

Tabela istine za binarni sabirac izgleda ovako,

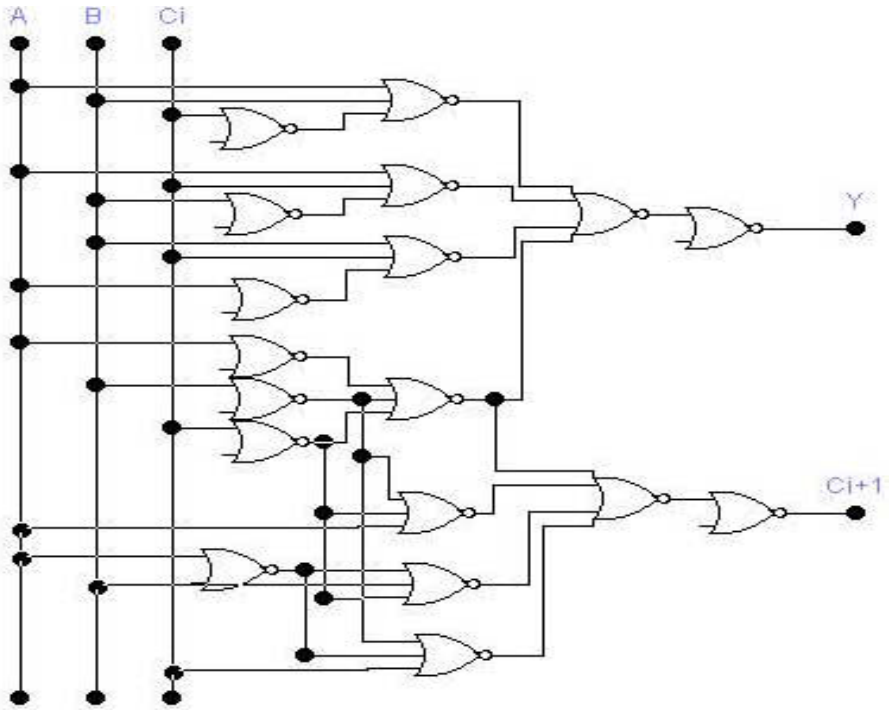
ULAZI			IZLAZI	
A	B	C_i	Y	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

odavde je,

$Y = A'B'C_i + A'BC_i' + AB'C_i' + ABC_i = (A+B)'C_i + (A+C_i)'B + (B+C_i)'A + (A' + B')'C_i =$
$(A + B + C_i')' + (A + B' + C_i) + (A' + B + C_i)' + (A' + B' + C_i') =$
$((A + B + C_i')' + (A + B' + C_i) + (A' + B + C_i)' + (A' + B' + C_i'))"$, te takodjer,

$C_{i+1} = A'BC_i + AB'C_i + ABC_i' + ABC_i = (A+B')'C_i + (A'+B)'C_i + (A'+B')'C_i' + (A'+B')' C_i=$
$(A + B' + C_i')' + (A' + B + C_i')' + (A' +B' + C_i)' + (A' + B' + C_i')' =$
$((A + B' + C_i')' + (A' + B + C_i')' + (A' +B' + C_i)' + (A' + B' + C_i')')"$.

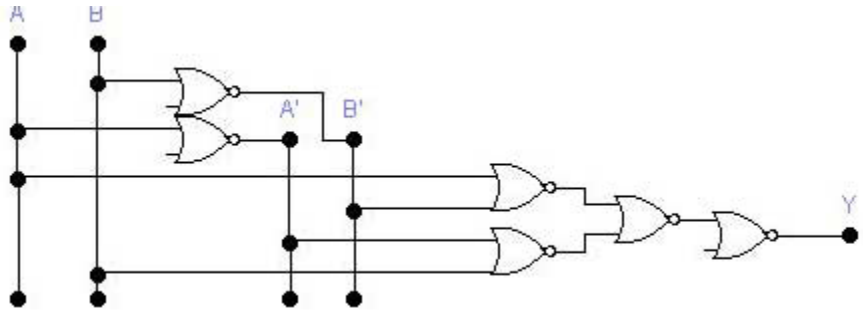
Dakle izrazili smo izlaze u funkciji ulaza ali u formatu pogodnom za realizaciju iskljucivo NILI kolima.U realizaciji to izgleda ovako,



Slika 3-5. Puni sabirac realizovan NILI kolima

Operacija ekskluzivno ILI (EXOR)

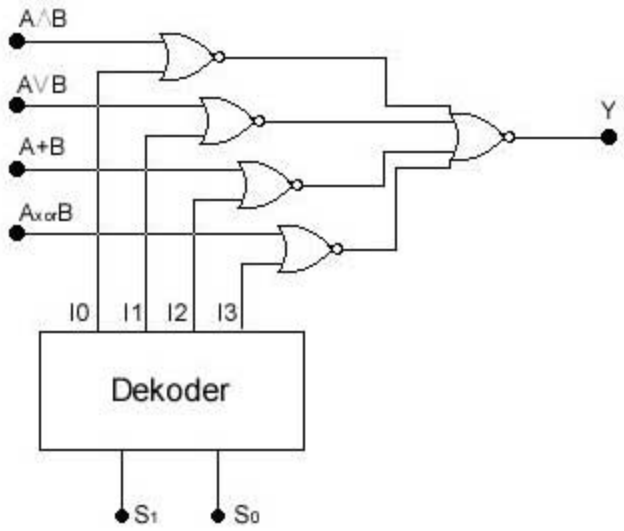
Ocito je $A_{xor}B = A'B + AB' = (A + B')' + (A' + B)' = ((A + B')' + (A' + B)')''$ što se NILI kolima realizuje na slijedeci nacin,



Slika 3-6. Ekskluzivno ILI realizovano NILI kolima

Dekoder

Dekoder je dio ALU zadudjen da na osnovu signala S_1 i S_0 propusti na izlaz samo jedan od rezultata operacija koje ALU modje izvesti.Blok dijagram i primjer modjemo vidjeti na slici 7.



Slika 3-7. Dekoder

Kako ovaj dekoder mora raditi? Na ulazima se pojavljuju 2 signala S_1 i S_0 .Dekoder mora izbaciti na svoje izlaze 3 jedinice i 1 nulu.Na izlazu NILI kola na koje dodju 1-ce imacemo nulu,a na ono NILI kolo koje otvaramo dovodimo nulu pa na njegovom izlazu imamo negiranu vrijednost ulaza npr. (A+B)' .Na izlazno NILI kolo sada dolaze 3 nule i (A+B)' te je Y=(A+B).

Formirajmo tabelu istine za ovakav dekoder (koji aktivira NILI kolo u nuli),

funkcija	**S_1**	**S_0**	**I_3**	**I_2**	**I_1**	**I_0**
A∧B	0	0	1	1	1	0
A∨B	0	1	1	1	0	1
A+B	1	0	1	0	1	1
$A_{xor}B$	1	1	0	1	1	1

Iz ove tabele se vidi da je,

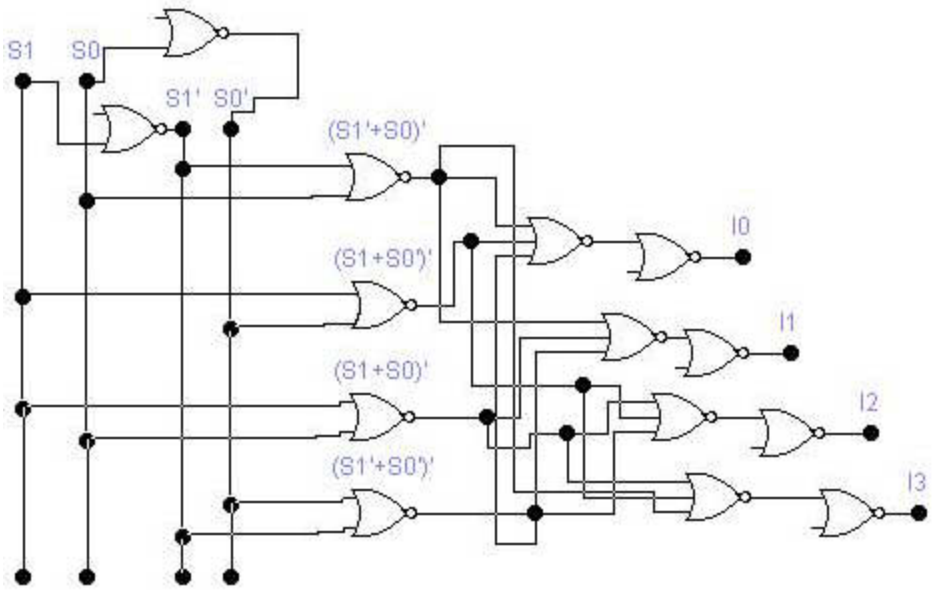
$I_0 = S_1'S_0 + S_1S_0' + S_1S_0 = (S_1 + S_0')' + (S_1' + S_0)' + (S_1' + S_0')' =$
$((S_1 + S_0')' + (S_1' + S_0)' + (S_1' + S_0')')''$

$I_1 = S_1'S_0' + S_1S_0' + S_1S_0 = (S_1 + S_0)' + (S_1' + S_0)' + (S_1' + S_0')' =$
$((S_1 + S_0)' + (S_1' + S_0)' + (S_1' + S_0')')''$

$I_2 = S_1'S_0' + S_1'S_0 + S_1S_0 = (S_1 + S_0)' + (S_1 + S_0')' + (S_1' + S_0')' =$
$((S_1 + S_0)' + (S_1 + S_0')' + (S_1' + S_0')')''$

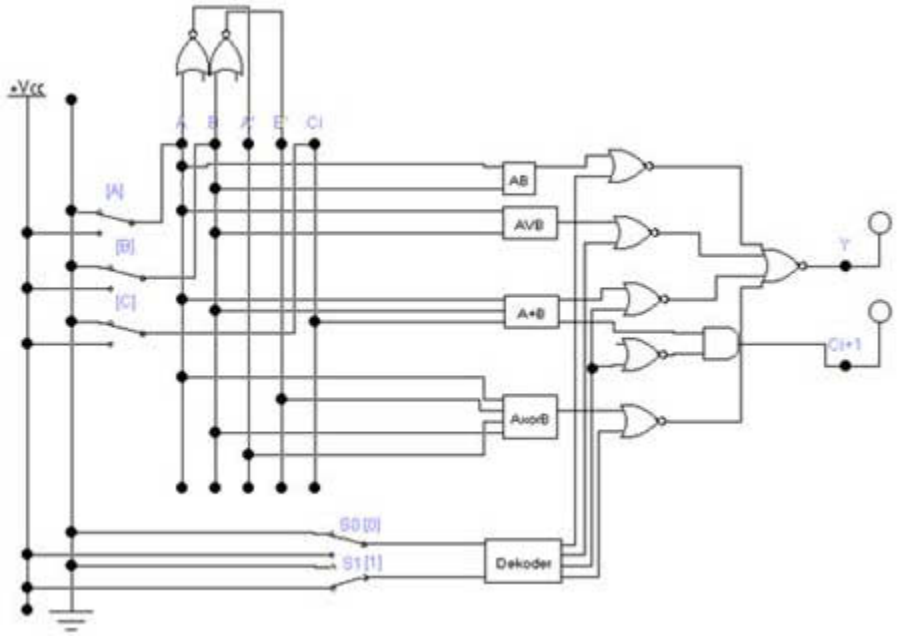
$I_3 = S_1'S_0' + S_1'S_0 + S_1S_0' = (S_1 + S_0)' + (S_1 + S_0')' + (S_1' + S_0)' =$
$((S_1 + S_0)' + (S_1 + S_0')' + (S_1' + S_0)')''$

Pa se dekoder na osnovu ovih izraza realizuje NILI kolima kao na slici na sljedecoj stranici.



Slika 3-8. Dekoder realizovan NILI kolima

Nakon što sam realizovao sve dijelove ALU naznacene na slici2. modjemo dati i konacni sklop. On nastaje jednostavnim uvezivanjem zajednickih linija svih posklopova,a konacnu realizaciju ALU modjete vidjeti na donjoj slici,



Slika 3 -9. ALU u fizickoj realizaciji

Modjete primjetiti kako ce izlaz C_{i+1} (izlaz za prenos u operaciji binarnog sabiranja) biti aktivan tek kada dekoder odredi da ALU vrši tu operaciju.

Ako kola imaju kašnjenje 5 ns tada je maximalno kašnjenje rezultata 30 ns jer podsklop punog sabiraca (koji je najkomplesniji) ima 4 nivoa logickih kola plus dva nivoa na izlazu to je ukupno 6*5ns = 30 ns.

Na kraju neke napomene o simulaciji,

Simulacija: prilodjena datoteka **alu-zadatak3.ewb** (kreirano u EWB v.5.0a)

A (tipka A) - ulaz
B (tipka B) - ulaz
C (tipka C) - ulazni prenos

Y sijalica - gori Y = 1 , ne gori Y=0
C_{i+1} sijalica - gori $C_{i+1} = 1$, ne gori $C_{i+1} = 0$

S1,S0 (tipke 1 i 0) - aktivira operaciju prema donjoj tabeli,

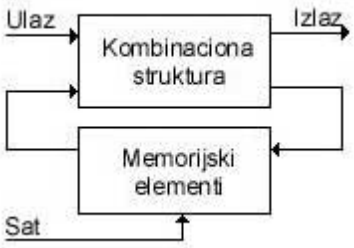
S1 tipka 1	**S0** tipka 0	**aktivira operaciju**
0	0	A i B
0	1	A ili B
1	0	sabiranje
1	1	A xor B

Napomena: Ako djelite vidjeti unutarnju imlementaciju pojedinih podsklopova - dvostruki klik mišem na taj podsklop.

Zadatak 4. Sinhrona sekvencijalna struktura ima 2^n mogucih stanja, j ulaznih i k izlaznih signala.Stanje na izlazima kao i naredno stanje varijabli stanja, zavise samo od trenog stanja varijabli stanja.Kako se ovakva struktura modje realizovati uz pomoc ROM-a? Koliki je minimalni kapacitet tog ROM-a?

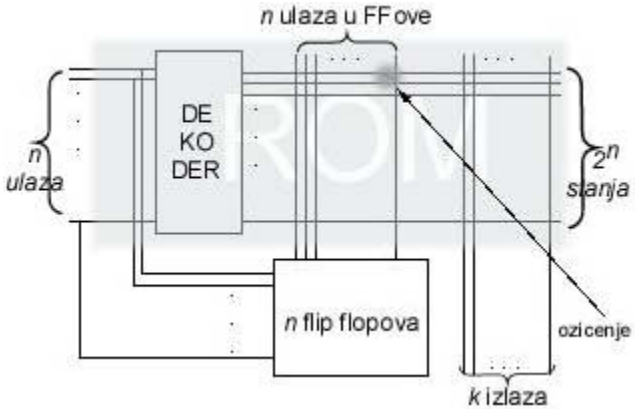
U ovom zadatku postoji nedoumica.Naime prvo se navodi da postoji " *j* ulaznih ...signala" a zatim da " ...stanje na izlazima kao i naredno stanje varijabli stanja, *zavise samo od trenutnog stanja varijabli stanja*" ali ne i od ulaznih signala.S toga cu ovaj zadatak riješiti u obje verzije.

a) ...stanje na izlazima kao i naredno stanje varijabli stanja, *zavise samo od trenutnog stanja varijabli stanja*
Model sinhrone sekvencijalne strukture izgleda kao na slici 4-1.Imamo jednu kombinacionu strukturu (po postavci zadatka to treba da bude ROM) i memorijski element (koji "cuva" stanje strukture).



Slika 4-1. Sinhrona sekvencijalna struktura

Kako po postavci zadatka, stanje na izlazima kao i stanje memorijskih elemenata ne zavise od ulaznog signala vec samo od trenutnog stanja memorijskih elemenata, to ovakva struktura realizovana pomocu ROM-a izgleda kao na slici 4-2.



Slika 4-2. Struktura realizovana pomocu ROM-a

Diskusija. ROM ima *n* ulaza koji idu direktno iz memorijskog elementa.Opet napominjem da nisam upotrijebio dodatnih *j* ulaza jer *k* izlaza i *n* ulaza u memorijski element ne zavise od tih dodatnih *j* ulaza.Kako ROM ima *n* ulaza to ima 2^n adresnih linija, a svaka adresna linija adresira *(n+k)bitni* podatak.Kako postoji 2^n adresnih linija to je moguce da postoji maximalno isto toliko stanja.
Minimalni kapacitet ovog ROM-a je prema tome,

$$\mathbf{K_{min} = (n+k)2^n \text{ bita.}}$$

Napomena : Ovakvom sekvencijalnom strukturom ne modjemo neposredno upravljati. Redoslijed stanja strukture direktno je odredjen odjicenjima u ROM-u.Prije ili kasnije struktura ce preci u stanje u kojem je vec ranije bila,pa ce se od tog trenutka struktura ponašati kao brojac, tj. prolazice kroz unaprijed definisanu sekvencu stanja.

b) ..stanje na izlazima kao i naredno stanje varijabli stanja, *zavise od trenutnog stanja varijabli stanja i od ulaznih signala*

U ovom slucaju postoji $n+j$ ulaza u ROM i prema tome 2^{n+j} adresnih linija.Kako u memorijski element vodimo

8 ulaza i dalje je moguce samo 2^n stanja varijabli stanja. Minimalni kapacitet je sada,

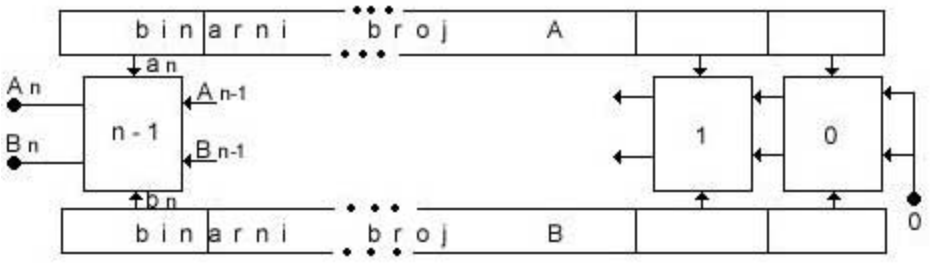
$$\mathbf{K_{min} = (n+k)2^{n+j}\ bita.}$$

Napomena: Ako upotrijebimo *j* ulaza onda ovom strukturom modjemo direktno upravljati, tj. odredjivati naredno stanje strukture što pod (a) nije bio slucaj.

Zadatak 5. Projektovati vremensku iterativnu strukturu za poredjenje binarnih brojeva u 2KK. Strukturu nacrtati i simulirati u Electronic WorkBenchu.

Projektovati cu strukturu za poredjenje n-bitnih binarnih brojeva predstavljenih u 2 KK. Za brojeve A i B vrijedi, ako je A>B tada je 2KK(A)<2KK(B) ,i obrnuto, što je s obzirom nacin nastanka 2KK (upotrijebljeno komlementiranje) sasvim ocito.

Kako ce izgledati taj komparator? Na donjoj slici vidimo n-bitni komparator,



Slika 5-1. N-bitni komparator

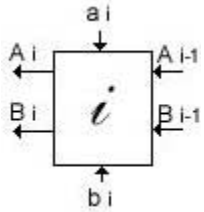
Sva kola oznacena rednim brojem 0,1,...,n-1 su istog dizajna i njihov zadatak je da uporedjuju odgovarajuce bite(cifre) binarnih brojeva A i B.

Za izlaze A i B mora vrijediti slijedeca tabela istine,

A	**B**	**Znacenje**
0	0	broj A = broj B
0	1	broj A < broj B
1	0	broj A > broj B
1	1	nemoguce stanje

Napomena: Pošto komparator poredi brojeve u 2KK kada ga isprojektujem na izlaze cu dodati dva invertora s obzirom da ako je 2KK(A)>2KK(B) tada je A<B.Zato je gornja tabela stanja privremena (koristim je dok ne isprojektujem Sliku 5-1) a kada na izlaze dodam dva invertora i tabela istine ce biti "invertovana".U svakom slucaju navesti cu je na kraju da modjete pratiti simulaciju.

Izdvojimo posebno 1-bitni komparator,on je prikazan na donjoj slici,



Slika 5-2. Jednobitni komparator

Formirajmo tabelu istine za jednobitni komparator,

Ai-1	Bi-1	ai	bi	Ai	Bi
0	0	0	0	0	0
0	0	0	1	0	1
0	0	1	0	1	0
0	0	1	1	0	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	1	0
0	1	1	1	0	1
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	1	0
1	0	1	1	1	0

*Napomena : Na ulazu u komparator ne modje biti 1 1 zato ova tabela ima 12 (a ne 16)vrsta.

Formirajmo Karnoove mape za svaki od izlaza;

Ai:

Ai-1Bi-1 \ aibi	00	01	11	10
00	0	0	0	1
01	0	0	0	1
11	X	X	X	X
10	1	0	1	1

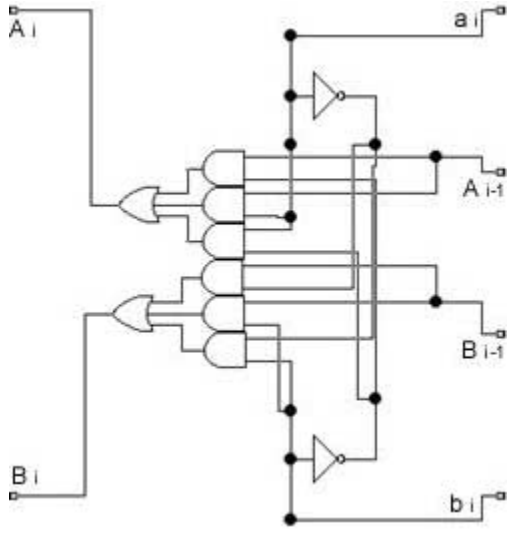
Bi:

Ai-1Bi-1 \ aibi	00	01	11	10
00	0	1	0	0
01	1	1	1	0
11	X	X	X	X
10	0	1	0	0

Odavde je,

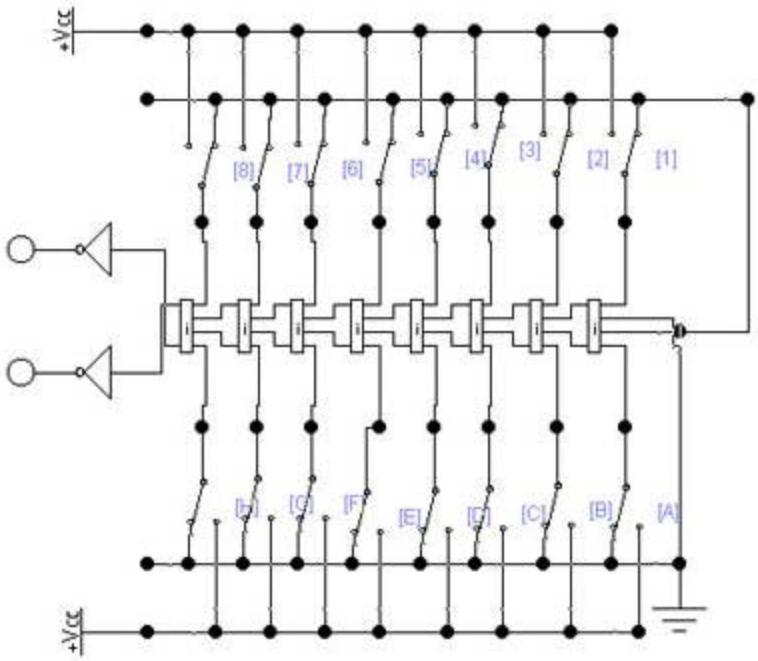
$A_i = A_{i-1}b_i' + A_{i-1}a_i + a_i b_i'$ $B_i = B_{i-1}a_i' + B_{i-1}b_i + a_i' b_i$

Na donjoj slici modjete vidjeti fizicku realizaciju gornjih jednacina,



Slika 5-3. Fizicka realizacija jednobitnog komparatora

Vezivanjem n jednobitnih komparatora dobijamo n-bitni komparator,ciju sliku modjete vidjeti na sledecoj stranici,



Slika 5-4. 8-bitni komparator

NAPOMENA:
Gornji red prekidaca simulira 8-bitni broj A,a donji red broj B.Kao što vidite na izlazima zadnjeg 1-bitnog komparatora nalaze se invertori koji vrše korekciju izlaza s obzirom da su brojevi u 2KK(u prethodnom tekstu sam pojasnio zašto).

TABELA ISTINE ZA SIMULACIJU:

gornja sijalica	**donja sijalica**	**znacenje**
0	0	nema struje
0	1	B>A
1	0	A>B
1	1	A=B

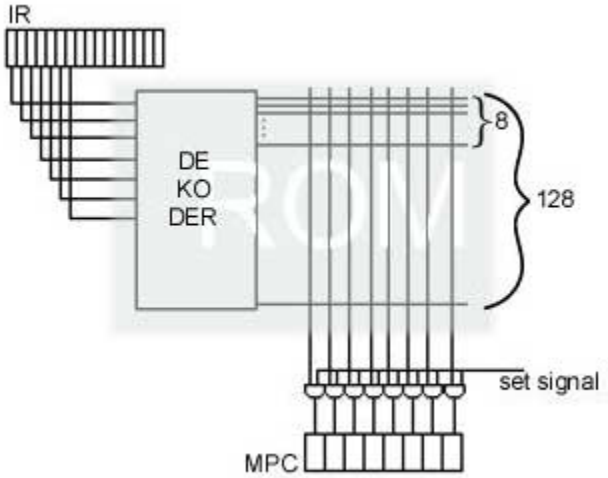
sijalica: 0 ne gori
1 gori

SIMULACIJA: datoteka **komparator_zadatak5.ewb** prilodjena uz rad
(1-bitni komparatori su u fajlu prikazani kao podkola,ako djelite vidjeti njihovu fizicku realizaciju – dvostruki klik mišom)

Zadatak 6. Kakve promjene na dizajnu oglednog procesora bi bile potrebne kada bi se iz mikroinstrukcije iskljucilo polje ADDR? Uporediti velicinu originalnog i novonastalog mikrokoda?

Za šta nama uošte sludji polje ADRESA u mikroinstrukciji? Sludji nam za grananje u mikroprogramu.To grananje, tj. prelazak na neku mikroinstrukciju koja nije slijedeca u nizu, koristimo kod (1) ispitivanja operacionog koda (makro)instrukcije, (2) povratak na pocetak mikroprograma i (3) korištenja istog "komada" mikrokoda. Ako iz mikroinstrukcije izbacimo polje ADRESA onda ocito na neki drugi nacin moramo riješiti navedene probleme.

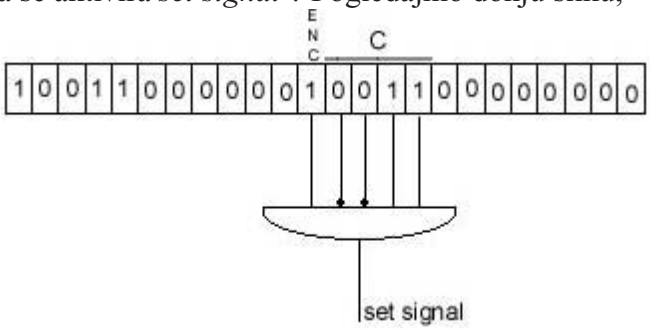
Prvi problem koji trebamo riješiti je dekodiranje operacionog koda (makro)instrukcije. Problem modjemo riješiti uvodjenjem hardwareskog dekodera koji se sastoji od jednog ROM-a u kojem su zapisane adrese pocetka rutine za interpretiranje pojedine instrukcije.Na slici vidimo realizovan dekoder operacionog koda,



Slika 6-1. Dekoder operacionog koda instrukcije realizovan pomocu ROM-a

S obzirom da imamo 15 instrukcija kod kojih su 4 najznacajnija bita operacioni kod i 8 instrukcija kod kojih su 7 najznacajnih bita operacioni kod, to koristim ROM sa 7 ulaza i 2^7=128 adresnih lokacija.
Instrukcija LODD ima operacioni kod 0000 , prema tome kombinacija 0000xxx na ulazu u dekoder ROM-a aktivirace jednu od prvih 8 adresnih linija. Zato na svih 8 adresnih linija mora biti pohranjen isti podatak tj. adresa rutine u Upravljackoj Memoriji koja ce interpretirati LODD instrukciju. Na ovaj nacin imamo realizovano 15 instrukcija * 8 adresnih linija =120 adresnih linija plus 8 adresnih linija na kojima se nalazi podatak o adresi rutine za obradu 8 instrukcija sa 7-bitnim operacionim kodom.Set signal sludji za sinhronizaciju.Ovaj signal je aktivan u 4.podfazi ciklusa clocka i tada propusti sadrdjaj iz ROM-a u mikroprogramski brojac (MPC).

Postavlja se pitanje kako i kada se aktivira *set signal* ? Pogledajmo donju sliku,



Slika 6-2. Aktiviranje set signala

Na gornjoj slici vidimo mikroinstrukciju *ir := mbr;* ovom instrukcijom smiješta se sadrdjaj memorijskog buffer registra (MBR) u instrukcijski registar (IR).Nakon ove instrukcije u oglednom mikroprogramu pocinje

dekodiranje operacionog koda.Ovo je takodjer jedino mjesto u mikroprogramu gdje se puni sadrdjaj IR-a.Zato nijedna druga instrukcija u poljima ENC i C ne sadrdji istovjetan sadrdjaj, te sam iz tog razloga ova polja upotrijebio za kreiranje set signala.
Dakle kada mikroprogram krene u dobavljanje nove instrukcije , ona ce biti smještena u IR. Time se na ulaz ROM-a sa slike 6-1 dovode novi signali koji ce na izlaz ROM-a izbaciti podatak koji predstavlja adresu pocetka rutine za izvršavanje te instrukcije.Ta adresa se u MPC ubacuje u 4-toj fazi Clocka kada se i obavlja dekodiranje instrukcije.U 1-voj fazi novog ciklusa Clocka sadrdjaj MPC-a ce se upotrijebiti za prebacivanje odgovarajuce mikroinstrukcije iz Upravljacke Memorije u mikroinstrukcijski registar (MIR).Set signal koji omogucava punjenje MPC-a aktivira se samo kada je u MIR-u instrukcija *ir:=mbr*.Time je obezbjedjeno da se u MPC ne upisuje sadrdjaj iz ROM-a u 4-toj fazi bilo kog Clocka vec samo onda kada to treba.

Ostalo nam je da riješimo problem skoka na pocetak mikroprograma,kao i problem instrukcija koje dijele zajednicki mikrokod.Oba problema riješicemo modifikacijom INKREMENTERA koji ce imati jedan ulazni RESET signal.Taj signal prouzrokuje postavljanje Inkrementera pa time i MPC-a na pocetak mikroprograma. Kada instrukcije koriste mikrokod druge instrukcije onda se na taj mikrokod prebacujemo, bezuslovno ili uslovno.Problem bezuslovnog grananja jednostavno riješavamo "kopiranjem" mikrokoda u rutinu za obradu tekuce instrukcije.
Npr. Problem bezuslovnog grananja kod ADDL instrukcije ciji mikrokod glasi,

13 rd;
14 ac:=mbr+ac;goto 0;//kraj interpretacije,skok na pocetak mikroprograma
.
.
.//pocetak interpretacije ADDL instrukcije
36 a:=ir+sp;
37 mar:=a;rd;goto 13; // bezuslovni skok

riješavamo tako što ce biti,
//pocetak interpretacije ADDL instrukcije-modifikovano
36 a:=ir+sp;
37 mar:=a;rd;
38 rd;
39 ac:=mbr+ac;reset;// kako se desi reset bice objašnjeno u nastavku teksta

Nešto komplikovaniji slucaj jeste uslovno grananje,ovdje cu modifikovati mikrokod tako da se skok modje desiti samo na pocetak mikroprograma – *reset* aktivan.Postoje 4 instrukcije koje koriste uslovno grananje:
JPOS

nemodifikovani mikrokod	modifikovani kod
21 alu:=ac;if n then goto 0; 22 pc:=band(ir,amask);goto0;	21 alu:=ac;if n then reset; 22 pc:=band(ir,amask);reset;

JZER

nemodifikovani mikrokod	modifikovani kod
22 pc:=band(ir,amask);goto0; 23 alu:=ac;if z then goto 22; 24 goto 0;	23 alu:=ac;if ***not z*** then reset; 22 pc:=band(ir,amask);reset;

JNEG

nemodifikovani mikrokod	modifikovani kod
22 pc:=band(ir,amask);goto0; 42 alu:=ac;if n then goto 22; 43 goto 0;	42 alu:=ac;if ***not n*** then reset; 43 pc:=band(ir,amask);reset;

JNZE

nemodifikovani mikrokod	modifikovani kod
44 alu:=ac;if z then goto 0; 45 pc:=band(ir,amask);goto0;	21 alu:=ac;if z then reset; 22 pc:=band(ir,amask);reset;

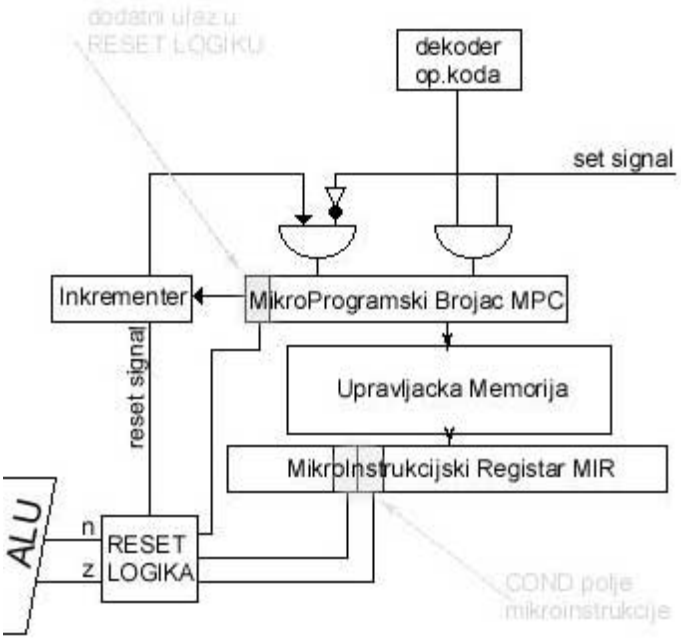
Modjemo primjetiti tri stvari:

1. nemodifikovani i modifikovani kod rade identic2. ne stvari,
3. modifikovani kod predvidja skok jedino na poc4. etak mikroprograma,
5. modifikovani kod ima dodatne uslove *if not n* i *if not z*.

Cinjenica 1. se podrazumjeva, 2. je pohvalna ali zato 3. nije jer sada imamo dodatna dva uslova pa nam 2 bita polja COND nece biti dovoljna da predstave,sada,6 mogucih uslova.
Naime potreban nam je dodatni ulaz na M-SEQ LOGIKU u daljem tekstu RESET LOGIKU da bi smo sva stanja predstavili,evo kako,

dodatni ulaz	**COND2**	**COND1**	**inkrementer**	
X	0	0	ne resetuj	
0	0	1	resetuj ako je n=1	if n then reset;
1	0	1	resetuj ako je z=0	if **not z** then reset;
0	1	0	resetuj ako je z=1	if z then reset;
1	1	0	resetuj ako je n=0	if **not n** then reset;
X	1	1	bezuslovni reset	reset;

Kako obezbjediti dodatni ulaz?
Jedno riješenje je proširiti COND polje mikroinstrukcije na 3 bita.Ovo medjutim daje 25-bitnu mikroinstrukciju pa bi nam trebao 25-bitni ROM u Upravljackoj memoriji.Kako je 25-bitni ROM krajnje nestandardan ovo riješenje necu upotrijebiti.
Drugi nacin je upotreba najznacajnijeg bita iz mikroprogramskog brojaca MPC.Pogledajmo sliku na slijedecoj stranici.



Slika 6-3. Riješenje problema uslovnog grananja

Ako na ovaj nacin obezbjedimo dodatni ulaz u RESET LOGIKU,jedino što treba dodatno podesiti je to da pocetak rutine za obrade instrukcija
JPOS i JNZE –budu smještene na adresi 0XXXXXXX u Upravljackoj Memoriji te,
JZER i JNEG – budu smještene na adresi 1XXXXXXX u Upravljackoj Memoriji.

Sada smo u fazi kada modjemo napisati kompletan modifikovani mikroprogram te dijagram modifikovanog oglednog procesora.

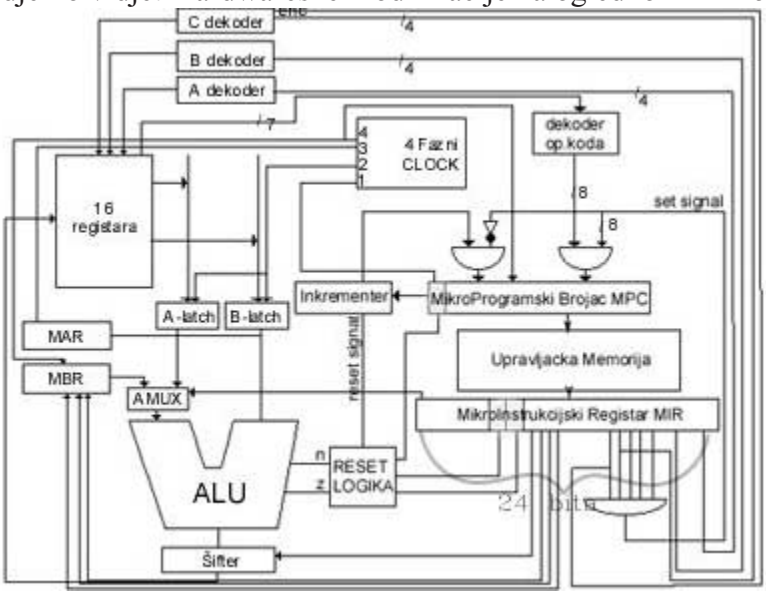
Modifikovani mikroprogram

0 mar:=pc;rd;
1 pc:=pc+1;rd;
2 ir:=mbr;
// rutina za LODD instrukciju
3 mar:=ir;rd;
4 rd;
5 ac:=mbr;reset;
// rutina za STOD instrukciju
6 mar:=ir;mbr:=ac;wr;
7 wr;reset;
// rutina za ADDD instrukciju
8 mar:=ir;rd;
9 rd;
10 ac:=mbr+ac;reset;
// rutina za SUBD instrukciju
11 mar:=ir;rd;
12 ac:=ac+1;rd;
13 a:=inv(mbr);
14 ac:=ac+a;reset;

```
// rutina za JPOS instrukciju
15 alu:=ac; if n then reset;
16 pc:=band(ir,amask);reset
// rutina za JUMP instrukciju
17 pc:=band(ir,amask);reset;
// rutina za LOCO instrukciju
18 ac:=band(ir,amask);reset;
// rutina za LODL instrukciju
19 a:=ir+sp;
20 mar:=a;rd;
21 rd;
22 ac:=mbr;reset;
// rutina za STOL instrukciju
23 a:=ir+sp;
24 mar:=a;mbr:=ac;wr;
25 wr;reset;
// rutina za ADDL instrukciju
26 a:=ir+sp;
27 mar:=a;rd;
28 rd;
29 ac:=mbr+ac;reset;
// rutina za SUBL instrukciju
30 a:=ir+sp;
31 mar:=a;rd;
32 ac:=ac+1;rd;
33 a:=inv(mbr);
34 ac:=ac+a;reset;
// rutina za JNZE instrukciju
35 alu:=ac;if z then reset;
36 pc:=band(ir,amask);reset;
// rutina za CALL instrukciju
37 sp:=sp+(-1);
38 mar:=sp;mbr:=pc;wr;
39 pc:=band(ir,amask);wr;reset;
// rutina za  PSHI instrukciju
40 mar:=ac;rd;
41 sp:=sp+(-1);rd;
42 mar:=sp;wr;
43 wr;reset;
// rutina za POPI instrukciju
44 mar:=sp;sp:=sp+(+1);rd;
45 rd;
46 mar:=ac;wr;
47 wr;reset;
// rutina za PUSH instrukciju
48 sp:=sp+(-1);
49 mar:=sp;mbr:=ac;wr;
50 wr;reset;
// rutina za POP instrukciju
51 mar:=sp;sp:=sp+(+1);rd;
52 rd;
```

53 ac:=mbr;reset;
// rutina za RETN instrukciju
54 mar:=sp;sp:=sp+(+1);rd;
55 rd;
56 pc:=mbr;reset;
// rutina za SWAP instrukciju
57 a:=ac;
58 ac:=sp;
59 sp:=a;reset;
// rutina za INSP instrukciju
60 a:=band(ir,smask);
61 sp:=sp+a;reset;
// rutina za DESP instrukciju
62 a:=band(ir,smask);
63 a:=inv(a);
64 a:=a+(+1);
65 sp:=sp+a;reset;
// rutina za JZER instrukciju
128 alu:=ac;if not z then reset;
129 pc:=band(ir,amask);reset;
// rutina za JNEG instrukciju
130 alu:=ac; if not n then reset;
131 pc:=band(ir,amask);reset;
//kraj mikroprograma

Primjetimo da su rutine za obradu instrukcija JZER i JNEG smještene na adrese 128 (10000000_2) i 131 (10000011_2) tako da je najznacajniji bit u MPC-u jedinica što je potrebno da bi RESET logika znala protumaciti dijelove *if not z* i *if not n* ovih mikroinstrukcija.
Na koncu originalni mikroprogram je imao 78 mikroinstrukcija a ovaj modifikovani ima 69 mikroinstrukcija,dakle ostvarili smo poboljšanje.
Konacno na donjoj slici modjemo vidjeti hardwareske modifikacije na oglednom mikroprocesoru,



Slika 6-4. Hardwareske modifikacije na oglednom procesoru

Zadatak 7. Keš sistem ima procenat promašaja 8%. Ako je vrijeme pristupa SRAM-a 15ns, a DRAM-a 60ns, koliko je efektivno vrijeme pristupa memoriji?

memorija	*vrijeme pristupa*	*ucestalost pristupa u procentima*
keš	15 ns	92%
centralna	60 ns	8%

Odavde je efektivno vrijeme pristupa memoriji,

$t_{ef.} = 15\ ns*0,92 + 60\ ns*0,08 = 13,8 + 4,8 = 18,6\ ns.$

Zadatak 8. Izracunati vrijeme izvršavanja svake od instrukcija oglednog procesora ako mu je frekvencija osnovnog signala sata 100MHz? Napisati proizvoljnu sekvencu od 10 instrukcija i na vremenu njenog izvršavanja odrediti ubrzanje koje bi se postiglo uvodjenjem hardverskog dekodiranja instrukcija.

Kako je frekvencija osnovnog signala sata 100 MHz to onda sat ima osnovni period T=10 ns.Za vrijeme trajanja tog perioda ogledni procesor izvrši jednu mikroinstrukciju.Sa druge strane izvršavanje jedne (makro)instrukcije podrazumjeva njeno dobavljanje u procesor, tumacenje operacinog koda i izvršavanje.Da bi se ovo izvelo potrebno je da se obavi nekoliko mikroinstrukcija.Uzecemo za primjer instrukciju STOD,da bi se ona izvršila potrebno je da se izvrše slijedece mikroinstrukcije iz mikroprograma,

Primjer: Izvršavanje instrukcije STOD kroz izvršavanje mikroinstrukcija.
//dobavljanje
0 mar:=pc; rd;
1 pc:=pc+1; rd;
2 ir:=mbr; rd;//dekodiranje//*if n then goto 28;*
3 tir:=lshift(ir+ir); if n then goto 19;
4 tir:=lshift(tir); if n then goto 11;
5 alu:=tir; if n then goto 9;
.
.
.
//izvršavanje
9 mar:=ir; mbr:=ac;wr;
10 wr; goto 0;
Iz ovoga zakljucujemo da je za izvršavanje instrukcije STOD potrebno da se izvrši 8 mikroinstrukcija pa ce ukupno vrijeme trajanja izvršavanja ove instrukcije biti t = 80 ns.

Neke instrukcije trebaju konstantnu dudjinu vremena za izvršavanje a neke instrukcije koje u sebi sadrdje uslovno grananje ce se izvršiti u dudjem ili kracem vremeskom periodu u zavisnosti od ispunjenosti uslova. U donjoj tabeli imamo naznaceno potrebno vrijeme izvršavanja svake instrukcije,

Naziv instrukcije	Vrijeme izvršavanja (u nanosekundama)
LODD	90
STOD	80
ADDD	90
SUBD	100

JPOS	75 *
JZER	80
JUMP	70
LOCO	70
LODL	100
STOL	90
ADDL	100
SUBL	110
JNEG	80
JNZE	80
CALL	90
PSHI	130
POPI	130
PUSH	120
POP	120
RETN	120
SWAP	120
INSP	110
DESP	120

* Instrukcija grananja – uzeto srednje vrijeme

Ako ogledni procesor modificiramo tako da se dekodiranje instrukcije vrši hardwareski tada iz mikroprograma modjemo izbaciti dio koda nadledjan za dekodiranje.Ako ponovo za primjer uzmemo instrukciju STOD tada cemo imati,

Instrukcija STOD softwareski dekodirana		**Instrukcija STOD hardwareski dekodirana**	
Mikrokod	Vrijeme izvršavanja	Mikrokod	Vrijeme izvršavanja
0 mar:=pc; rd; 1 pc:=pc+1; rd; 2 ir:=mbr; rd;if n then goto 28; 3 tir:=lshift(ir+ir); if n then goto 19; 4 tir:=lshift(tir); if n then goto 11; 5 alu:=tir; if n then goto 9; . . . 9 mar:=ir; mbr:=ac;wr; 10 wr; goto 0;			
80 ns		*0 mar:=pc; rd; 1 pc:=pc+1; rd; 2 ir:=mbr; rd 3 mar:=ir;mbr:=ac; wr; 4 wr; goto 0;*	
		50 ns	

Dakle cim se sadrdjaj upiše u IR nastupa hardwersko dekodiranje i taj dekoder puni adresu memorijske lokacije iz Upravljacke Memorije (na kojoj se nalazi rutina za obradu STOD instrukcije) u MPC, tj. nisu potrebne mikroinstrukcije i polje ADRESA u mikroinstrukciji da to obave.

Ubrzanje postignuto hardwarskim dekodiranjem instrukcije STOD je 30 ns.U procentima u odnosu na softwaresko dekodiranje to je 80 – (80/100)*X=50 slijedi X=37,5%.

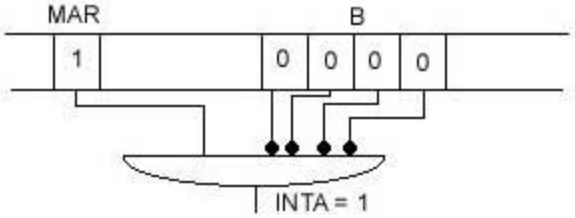
Uzmimo sekvencu od 10 instrukcija i izracunajmo ubrzanje koje smo dobili hardwareskim dekodiranjem instrukcija,

Sekvenca instr. sa soft dekodiranjem		**Sekvenca instr. sa hard dekodiranjem**	
Instrukcija	Vrijeme izvršavanja	Instrukcije	Vrijeme izvršavanja
0 LODD xx; ADDD xx; SUBD xx; JZER 1; SUBD xx; JZER 1; SUBD xx; JZER 1; JUMP 0; 1 JUMP 0;			
90 90 100 80 100 80 100 80 70 70		0 LODD xx; ADDD xx; SUBD xx; JZER 1; SUBD xx; JZER 1;	
SUBD xx; JZER 1; JUMP 0; 1 JUMP 0;		60 60 70 50 70 50 70 50 40 40	
Ukupno vrijeme:	860	Ukupno vrijeme:	560
Ukupno ubzanje: 860-(860/100)X=560 slijedi		**UBRZANJE = 34,88 %**	

Zadatak 9. Modifikovati ogledni procesor tako da podržava sistem prekida.

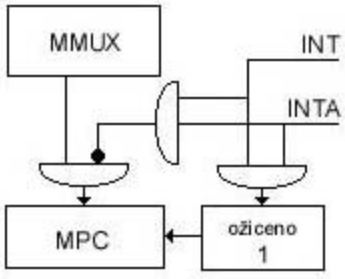
Smisao prekida je u tome da procesor radi svoj posao i zaustavi se da opsludji neku vanjsku jedinicu samo onda kada ta vanjska jedinica to i zatradji od procesora.Ja ovdje necu ulaziti u to kako je taj signal generisan i kako je dospio do procesora, vec kako modifikovati ogledni procesor da primi Interrupt zahtjev i da ga opsludji. Dakle naš ogledni procesor mora imati još jedan (asinhroni) ulaz INT koji ce "obavjestiti" procesor da je nastao prekid i da procesor mora da reaguje.Procesor ce reagovati na sledeci nacin:
- završiti ce izvršavanje tekuce (makro)instrukcije,
- pohranice sadržaj programskog brojaca PC na neko sigurno mjesto,
- kreirace signal INTA koji aktivira pocetak obrade prekida.

Kako obezbjediti da procesor nastavi svoj posao dok ne završi izvršavanje tekuce instrukcije?
Tako što cemo signal INTA koji aktivira pocetak obrade prekida generisati u momentu izvršavanja prve mikroinstrukcije mikroprograma.Kao što znamo to je instrukcija
0 mar:=pc; U mikroinstrukcijskom registru je stanje kao na donjoj slici,



Slika 9-1. Stanje MIR-a na pocetku obrade nove instrukcije

Iskoristicemo cinjenice da se mikroinstrukcija **0 mar:=pc;** izvršava samo na jednom mjestu u mikroprogramu,da se izvršava neposredno nakon završetka prethodne a prije pocetka naredne instrukcije i da se u programskom brojacu još nalazi adresa memorijske lokacije na kojoj se nalazi slijedeca instrukcija koja treba da se izvrši.
Kao što vidimo signal INTA se aktivira svaki put prije pocetka dekodiranja nove instrukcije a programski brojac nije inkrementiran.Da bi signal INTA mogao nešto da pokrene prethodno mora biti aktivan signal INT.Razmotrimo stanje na slijedecoj slici,



Slika 9-2. INT i INTA

Signal INT sam nemodje ništa da u radi.Tek kada se u pogodnom trenutku aktivira i INTA pojavice se nula na ulaznom AND gate-u u MPC pa u MPC nece biti upisana adresa preko MMUX-a.Umjesto toga imamo jedan 8-bitni registar sa odjicenim jedinicama koji ce nakon što ga ulazni AND gate aktivira (enable) napuniti svoj sadrdjaj u MPC.
Na adresi 255 (11111111_2) u upravljackoj memoriji nalazi se prva instrukcija mikrorutine za obradu prekida. Dakle i ROM upravljacke memorije treba modifikovati.Na toj lokaciji se nalazi sledeca mikroinstrukcija (zapisana u svom 32-bitnom formatu),

255 f:=pc; // Spasi sadrdjaj PC-a u registar F

COND polje ove mikroinstrukcije mora biti 11 a u polje ADRESA mora biti zapisana adresa slijedece mikroinstrukcije ove mikrorutine za obradu prekida.Napomenimo da se u momentu upisa instrukcije sa lokacije 255 u MIR signal INTA deaktivira pa je moguc skok na tu novu adresu.
Naredna mikroinstrukcija npr.smještena na liniji 254 je,

254 pc:=band(+1,amask); // Smjesti u programski brojac adresu #0FFF

Zašto koristim ovu mikroinstrukciju? Naime ja pretpostavljam da ovaj racunarski sistem radi na sledecem principu:

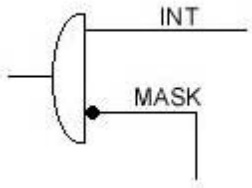
- vanjska jedinica (bilo koja) modje biti opsludjena na više nacina, i za svaki specificni problem postoji odgovarajuca rutina koja riješava taj problem,
- kada vanjska jedinica zatradji od procesora da nešto odradi ona mora znati koju to rutinu procesor treba izvršiti,
- vanjska jedinica na adresi #0FFF postavi instrukciju JUMP #XXXX; pri cemu je #XXXX pocetna adresa neke jedinstvene rutine,
- vanjska jedinica postavlja INT na ulaz procesora.

Za instrukciju sa linije 254 vrijedi : COND polje je u 11 (bezuslovni skok) a ADRESA je #00. Time se skace na pocetak mikroprograma koji ce uzeti instrukciju JUMP sa adrese #0FFF i nastaviti da radi svoj posao.

Mikroinstrukcija sa linije 254 je takodjer jedinstvena u mikroprogramu i njena polja ALU =1 , ENC=1,C=0 (ova kombinacija ne postoji ni u jednoj drugoj mikroinstrukciji) treba iskoristiti da se iskljuci INT signal.Signal INT ce biti maskiran (onesposobljen) sve do se ne izvrši mikroinstrukcija

253 pc:=f; // postavlja sadrdjaj iz registra F u PC

kao na donjoj slici,



Slika 9-3. Maskiranje interrupta instrukcijom sa linije 254

Kao što mikroinstrukcija 254 maskira tako ce mikroinstrukcija sa linije 253 demaskirati INT signal. Mikroinstrukcija sa linije 253 ima COND polje 11 a polje ADRESA je #00.Da bi uopce mogli koristiti ovu mikroinstrukciju mora biti definisana nova makroinstrukcija RETURN koja ce se nalaziti na kraju svake rutine za obradu prekida.To je 16-bitna instrukcija ciji je kod,

Binarni kod	Mnemonik	Instrukcija	Znacenje
1111 1111 0000 0000	RETR	vrati se u program	pc:=f;

U skladu sa ovim potrebno je dodatno modifikovati mikroprogram,

```
75 sp:=sp+a;goto 0;
76 tir := lshift (tir);if n then goto 253;          //   1111 1110 ili 1111 1111 ?
```

```
77 a:=band(ir,smask);                    // 1111 1110 DESP instrukcija
78 a:=inv (a);
79 a:=a+(+1);goto 75;
.
.
.
253 pc:=f;goto 0;                        // 1111 1111 RETR instrukcija
```

REZIME: Sistem prekida radi ovako:

1. Vanjska jedinica koja tradji prekid postavlja instrukciju JUMP #XXXX;na za tu svrhu rezervisanu adresu #0FFF u glavnoj memoriji, adresa #XXXX je adresa poc2. etka rutine za obradu prekida,
3. Vanjska jedinica postavlja INT signal procesoru,
4. Procesor završava tekuc5. u instrukciju i prije poc6. etka nove podidje INTA signal,
7. INT i INTA zajedno pune MPC adresom #FF na kojoj se nalazi prva mikroinstrukcija mikrorutine za obradu prekida,
8. Mikroinstrukcijom sa adrese #FF pohrani se sadrdjaj PC-a u registar F,i predje se na instrukciju #FE (signal INTA je skinut upisom instrukcije sa #FF u MIR),
9. Mikroinstrukcijom sa adrese #FE puni se PC adresom #0FFF,maskira se INT signal,
10. Procesor c11. ita JUMP #XXXX; instruciju sa lokacije #0FFF u glavnoj memoriji i izvršava odgovarajuc12. u rutinu(koja poc13. inje na adresi #XXXX),
14. Rutina (svaka) se obavezno završava instrukcijom RETR tako da procesor nakon dekodiranja operacionog koda ove instrukcije izvrši mikroinstrukciju sa linije #FD (253_{10}) koja u programski brojac15. PC napuni sadrdjaj registra F, demaskira INT signal, te skac16. e na poc17. etak mikroprograma,
18. Procesor nastavlja sa izvršenjem prekinutog glavnog programa.

Da bi sve ovo funkcionisalo nijedna rutina za obradu prekida NE SMIJE raditi sa registrom F, to je ogranicenje pri programiranju ovih rutina.
U ovoj realizaciji nije dozvoljeno prekidanje prekida.Da bi to realizovali potrebno je sadrdjaj PC pohranjivati na stek.(Nadjalost,zbog nedostatka vremena ne mogu to realizovati.)

Bajramovic Kemal
Hadjici, 03.09.2000.god.