POGLAVLJE 8.
PROJEKTOVANJE PROCESORA

U ovom poglavlju se ilustruju metode projektovanja sastavnih delova svakog procesora opšte namene. Obuhvaćeno je projektovanje aritmetičko logičke jedinice, upravljačke jedinice, registara i prenosnih puteva u okviru procesora. Pored toga, završni zadaci ovog poglavlja prikazuju postupak realizacije kompletnog procesora kao složenog digitalnog sistema. Hijerarhijski metod projektovanja digitalnih sistema se intenzivno koristi kroz većinu zadataka sa jasno izraženom dekompozicijom digitalnog sistema koji se projektuje.

Tipičan procesor se sastoji iz sledećih sastavnih delova:

- aritmetičko logičke jedinice,
- upravljačke jedinice,
- skupa registara i
- prenosnih puteva.

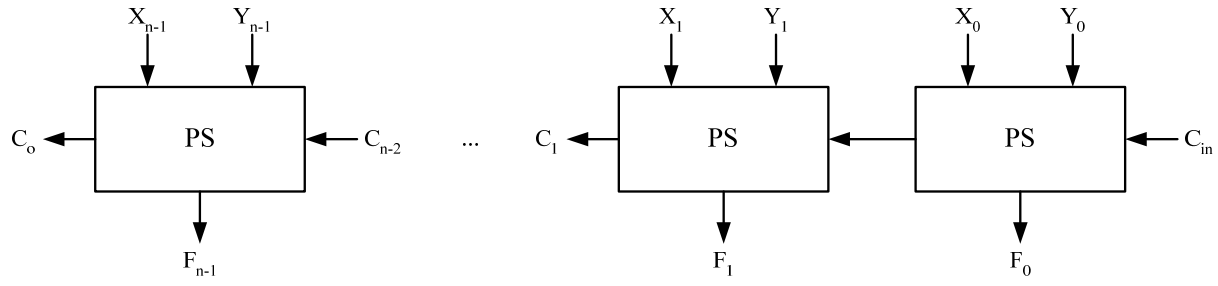
Zahvaljujući regularnoj strukturi sastavnih delova procesora omogućeno je njihovo nezavisno projektovanje.

PROJEKTOVANJE ARITMETIČKO LOGIČKE JEDINICE PROCESORA

Aritmetičko logička jedinica je zadužena za izvršenje aritmetičkih i logičkih operacija. Tu su sadržane operacije sabiranja, oduzimanja, množenja, delenja, promene znaka, upoređenje operanada i slično. Od logičkih operacija obično su podržane osnovne logičke operacije I, ILI, NE i kod većine procesora i operacija ekskluzivno ILI. Uobičajeno je da se u sintezi Aritmetičko Logičke Jedinice (ALJ) kreće se od sinteze punog sabirača (PS).

Projektovanje aritmetičko logičke jedinice kao kombinacione mreže se bazira na paralelnom sabiraču, Slika 8.1. Paralelni sabirač vrši operacije nad svim bitima operanada istovremeno, pa je broj jednorazrednih punih sabirača jednak broju razreda brojeva koji se sabiraju.

Rezultat aritmetičkih operacija nad bitima manje težine direkno utiče na vrednost bita veće težine preko signala izlaznog prenosa kao $C_{i(i)} = C_{o(i-1)}$, dok kod logičkih operacija ovog prenosa nema ($C_i = 0$).



Slika 8.1: Blok šema paralelnog sabirača sa n razreda

Logičke operacije ostvaruju se zabranom svih ulaznih prenosa u pun sabirač i dovođenjem odgovarajućih oblika ulaza.

Pored pristupa projektovanju ALJ preko kombinacionih mreža, postoji i pristup realizacije aritmetičkih operacija sa akumulatorom. Ovakav pristup se koristi kod realizacije složenih aritmetičkih operacija (množenje, delenje, korenovanje i sl.) koje zahtevaju iterativni postupak izračunavanja. Složene aritmetičke operacije se takođe mogu projektovati preko kombinacionih mreža. Međutim, ovakav pristup proizvodi veoma složene kombinacione mreže sa izuzetno dugim prenosnim putevima. Time se značajno redukuje maksimalna brzina izvršenja operacije. U praksi se od složenih aritmetičkih operacija jedino množenje (i u retkim slučajevima delenje) realizuje preko kombinacionih mreža.

U ovom poglavlju su ilustrovani postupci projektovanja množača i delitelja celih označenih i neoznačenih brojeva. Prikazan je kompletan postupak realizacije digitalnog sistema sa akumulatorom od zadatog algoritma, preko identifikacije osnovnih elemenata sistema i međusobnih veza do implementacije istog u VHDL-u pomoću hijerarhijskog pristupa projektovanju. Pored ilustracije hijerarhijskog projektovanja u VHDL-u, u ovom poglavlju se prikazuje i da su tok podataka i tok upravljačkih signala dva nezavisna dela svakog digitalnog sistema.

PROJEKTOVANJE UPRAVLJAČKE JEDINICE PROCESORA

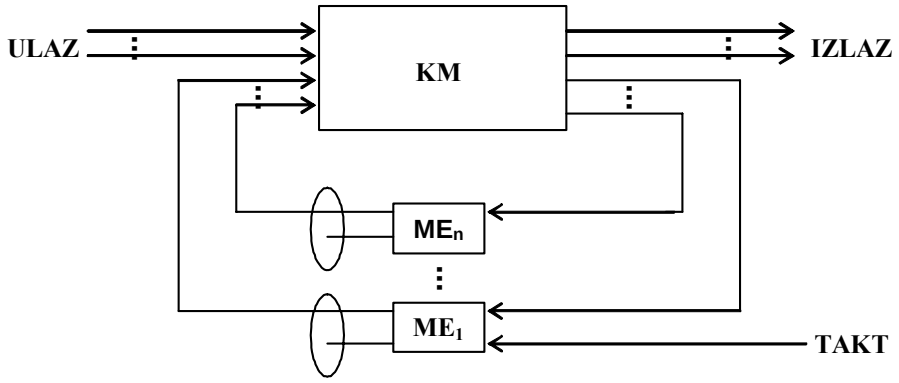
Upravljačka jedinica definiše vremenski redosled izvršenja pojedinih mikrooperacija u objektu upravljanja. Objekat upravljanja može biti na primer aritmetičko logička jedinica ili procesor. U tom slučaju upravljačka jedinica definiše sekvencu selekcionih promenljivih kojima se vrši izbor aritmetičke ili logičke operacije, adrese polaznih registara koji sadrže operande (signale dozvole čitanja tih registara) i adresu odredišnog registra gde se smešta rezultat izvršene operacije (signal dozvole upisa u registar).

Ilustrovane metode realizacije upravljačkih jedinica u ovom poglavlju se mogu grupisati u dva pristupa:

1. TRADICIONALNE metode – kojima se realizuju upravljački automati fiksne strukture i
2. MIKROPROGRAMSKI metod – kojim se realizuje upravljački automat sa programabilnom strukturom

Metod dodele jednom stanju jednog memorijskog elementa

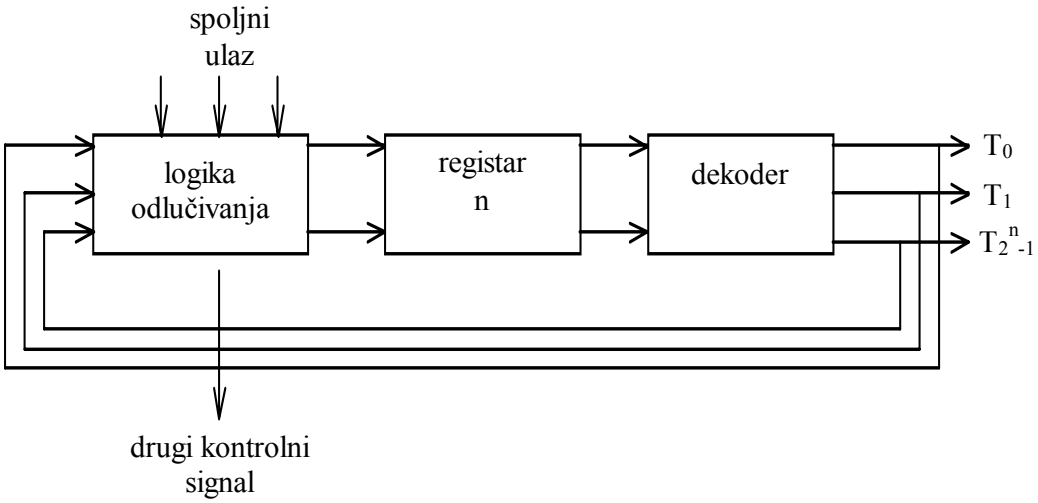
Po ovom metodu svakom stanju digitalnog sistema dodeljen je po jedan flip-flop, a u jedinici vremena samo je jedan od njih u stanju 1. Ovaj metod ne koristi minimalan broj memorijskih elemenata, pa se radi preglednosti šeme i manjeg broja ulaza najčešće koriste D flip-flopovi. Strukturnu šemu UJ prikazuje Slika 8.2.



Slika 8.2: Strukturna šema UJ

Realizacija UJ pomoću dekodera i registra sekvence

Realizacija UJ dekoderom i registrom sekvence podrazumeva kodiranje svakog stanja. Registar sekvence dužine n bita sadrži kod trenutno aktivnog stanja koji se dovodi na ulaz dekodera. Dekoder daje na izlazu 2^n međusobno isključivih stanja. Strukturnu šemu ove UJ prikazuje Slika 8.3.

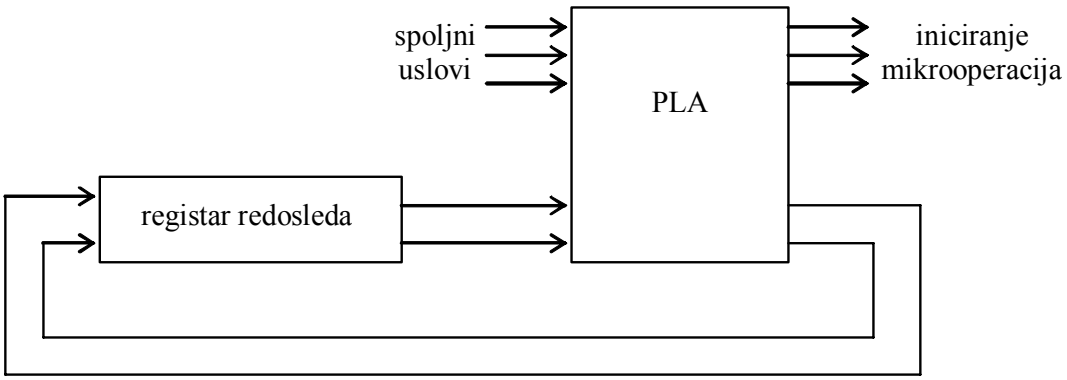


Slika 8.3: Strukturna šema UJ na bazi registra i dekodera

Realizacija UJ pomoću PLA i registra

Sinteza UJ pomoću PLA podrazumeva realizaciju kombinacionog dela UJ pomoću PLA. Osnovna prednost ove metode je smanjenje broja integralnih kućišta. Broj kućišta se svodi bilo na dva (PLA i registar), bilo samo na jedno kućište (kada je registar unutar samog PLA). Strukturnu šemu ove UJ prikazuje Slika 8.4.

Registar redosleda definiše tekuće stanje upravljačke jedinice. Deo izlaza iz PLA određuje koja će mikrooperacija biti započeta zavisno od spoljnih uslova i tekućeg stanja registra sekvence. U isto vreme drugi deo izlaza iz PLA definiše naredno stanje registra sekvence.

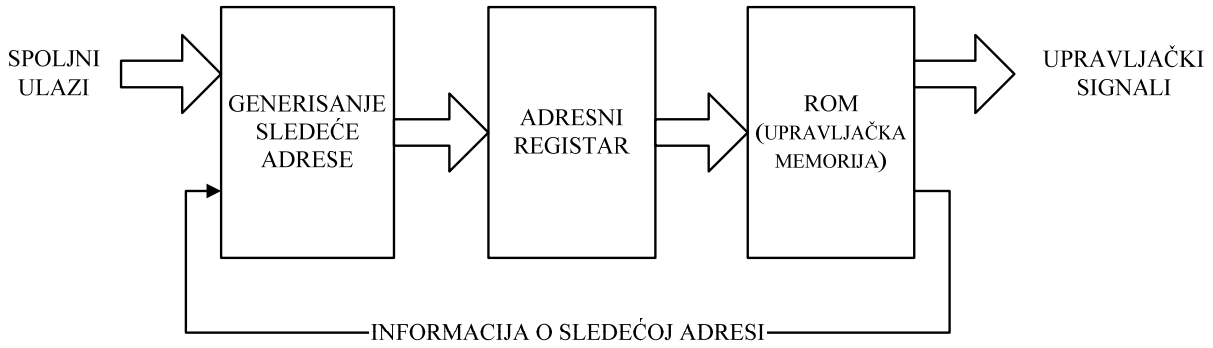


Slika 8.4: Strukturna šema upravljačke jedinice sa PLA

Mikroprogramska UJ

U opštem slučaju strukturu mikroprogramske upravljačke jedinice čine tri osnovne funkcionalne jedinice:

- memorija upravljačkih reči (upravljačka memorija)
- adresni registar
- generator adresa memorije upravljačkih reči.



Slika 8.5: Struktura mikroprogramske UJ

Generator sledeće adrese se sastoji od multipleksera 2×1, gde je jedna informacija sledeća adresa generisana od strane upravljačke memorije, a druga informacija početna adresa mikroprograma. Na osnovu spoljnih ulaza se vrši odluka koji podatak će biti prosleđen na izlaz multipleksera. Pored multipleksera u generatoru sledeće adrese se nalazi i kolo za upravljanje adresnim registrom.

Adresni registar je u principu jedan binarni brojač sa paralenim unosom. Generator sledeće adrese na osnovu spoljnih ulaza i informacije o sledećoj adresi vrši odluku da li će se u adresni registar upisati novi sadržaj (skok na neku adresu u mikroprogramu) ili će se sadržaj adresnog registra uvećati za jedan (sekvencijalno izvršavanje mikroinstrukcija).

REGISTRI PROCESORA

Pored aritmetičko logičke jedinice i upravljačke jedinice, procesor sadrži i skup registara sa jasno definisanom namenom. U ovaj skup spadaju sledeći registri:

- Programski brojač,
- Registar instrukcija,
- Memorijski adresni registar,
- Memorijski registar za prihvat podataka (Memorijski bazni registar),
- Ukazivač steka i
- Akumulatorski registar.

Kroz rešene zadatke u ovom poglavlju se ilustruje uloga svih navedenih registara u okviru procesora opšte namene.

PRENOSNI PUTEVI PROCESORA

Realizacija mikrooperacija u procesoru se svodi na razmenu sadržaja između registara. Prenosni putevi procesora omogućuju ovu razmenu sadržaja registara sa ciljem izvršenja tražene mikrooperacije.

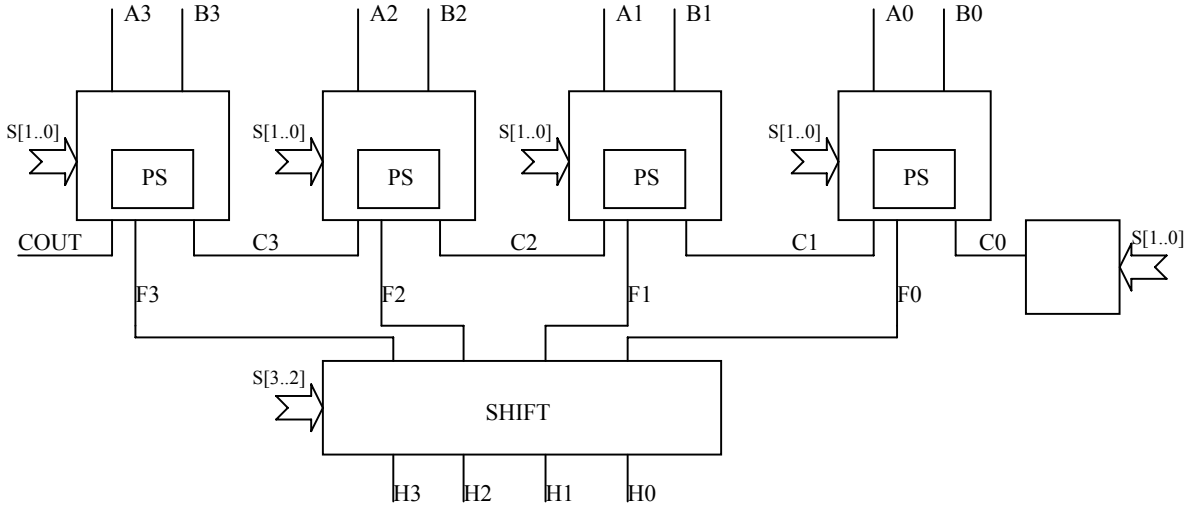
Postoje dva načina organizacije procesorskih prenosnih puteva. U prvom slučaju registri procesora se organizuju u obliku memorije. Tada se razmena sadržaja između registara realizuje preko memorijskih registara za prihvat podataka i memorijskih adresnih registara. Organizacija prenosnih puteva procesora preko magistrala predstavlja drugi način organizacije registara procesora. Kod ove metode se svaki procesorski registar tretira i kao polazni registar i kao odredišni registar. Tok podataka prilikom razmene sadržaja registara se kontroliše primenom multipleksera pomoću kojih se više polaznih registara svodi na jedinstveni prenosni put do odredišta.

U ovom poglavlju se kroz rešene zadatke ilustruju oba načina organizacije prenosnih puteva.

PROJEKTOVANJE ARITMETIČKO LOGIČKE JEDINICE PROCESORA

8.1 ZADATAK:

Izvršiti sintezu 4 bitne ALJ za operacije koje daje Tabela 8.1. ALJ sintetizovati pomoću NI kola. Voditi računa o labelama. Blok dijagram ALJ prikazuje Slika 8.6.



Slika 8.6: Blok šema ALJ koju treba sintetizovati

	S_1	S_0
A + B	0	0
A - B	0	1
$A \vee B$	1	0
$A \wedge B$	1	1

	S_3	S_2
ZERO	0	0
shl F	0	1
shr F	1	0
propuštanje	1	1

Tabela 8.1: Kodovi operacija koje treba da realizuje ALJ

REŠENJE:

Sinteza ALJ počinje definisanjem prenosnih funkcija punog sabirača. Poznavanjem funkcije punog sabirača, kreiranjem tablice ulaza/izlaza punog sabirača i minimizacijom izlaznih funkcija dobijaju se sledeće jednačine:

$$F = X_i \cdot \overline{Y_i} \cdot \overline{C_{in}} + \overline{X_i} \cdot \overline{Y_i} \cdot C_{in} + X_i \cdot Y_i \cdot C_{in} + \overline{X_i} \cdot Y_i \cdot \overline{C_{in}}$$

$$C_{out} = X_i \cdot C_{in} + Y_i \cdot C_{in} + X_i \cdot Y_i$$

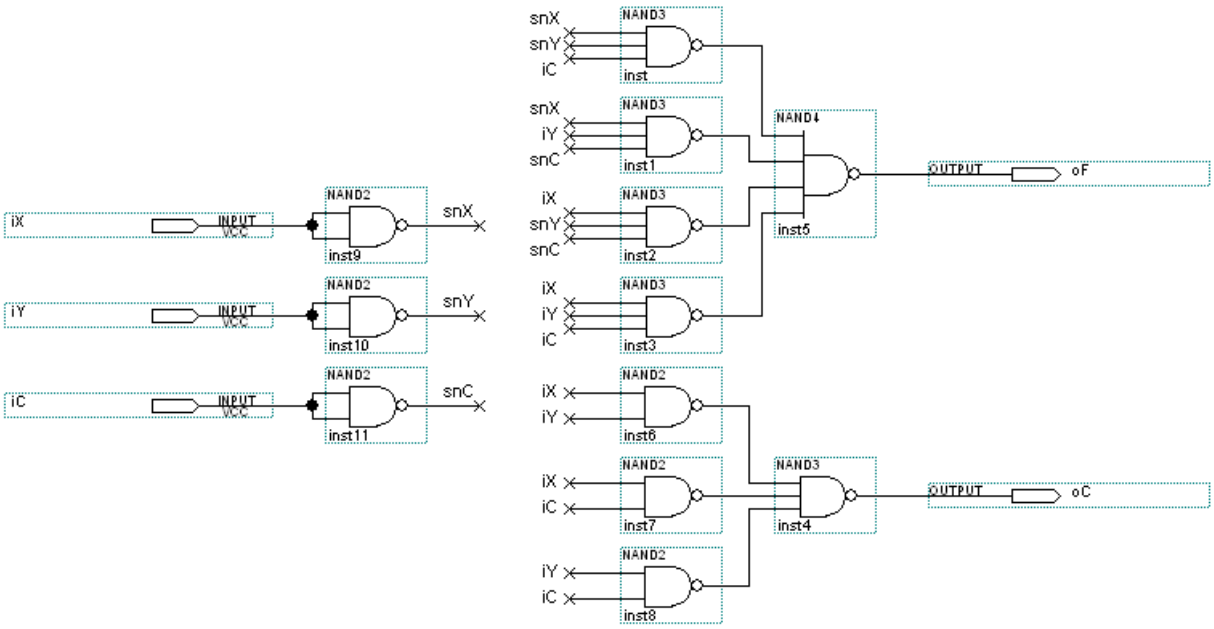
gde su ulazi punog sabirača označeni sa X_i, Y_i, ulazni prenos je označen sa C_{in}, dok je sa F označen izlaz iz punog sabirača, a izlazni prenos punog sabirača označen sa C_{out}.

Ove funkcije se korišćenjem De Morganovih zakona mogu prilagoditi u oblik pogodan za realizaciju sa NI logičkim kolima.

$$F \quad \overline{\overline{X_i \cdot \overline{Y_i} \cdot \overline{C_{in}}} \cdot \overline{\overline{X_i} \cdot \overline{Y_i} \cdot C_{in}} \cdot \overline{X_i \cdot Y_i \cdot C_{in}} \cdot \overline{\overline{X_i} \cdot Y_i \cdot \overline{C_{in}}}}$$

$$C_{out} = \overline{\overline{X_i \cdot C_{in}} \cdot \overline{Y_i \cdot C_{in}} \cdot \overline{X_i \cdot Y_i}}$$

Slika 8.7 prikazuje realizaciju punog sabirača pomoću NI logičkih kola.



Slika 8.7: Realizacija punog sabirača pomoću NI kola

Na osnovu prenosne funkcije punog sabirača, kao i na osnovu činjenice da se oduzimanje može realizovati kao sabiranje sa drugim komplementom:

$$A - B = A + \overline{B} + 1$$

formira se tabela koja prikazuje šta treba dovesti na ulaze punog sabirača da bi se realizovale pojedine operacije zadate u tekstu zadataka. Pri tome će se iskoristiti pretpostavka da se na ulaz X_i dovodi A_i, dok se ulazi Y_i moraju prilagoditi i to na sledeći način:

- za sabiranje se dovodi B_i
- za oduzimanje se dovodi $\overline{B_i}$

Za realizaciju logičkih operacija (I i ILI) prenos C_{in} se spaja na 0 jer kod logičkih operacija nema uticaja rezultata sa bita niže težine na bit više težine. Tada prenosna funkcija punog sabirača izgleda ovako:

$$F_i = A_i \oplus B_i = \overline{A_i} \cdot B_i + A_i \cdot \overline{B_i}$$

Uvažavajući pretpostavku da je na ulazu X_i=A_i, i ukoliko se želi da realizovati logička operaciju I (KONJUKCIJA) postavlja se sledeća jednačina:

$$F_i = A_i \cdot B_i = A_i \oplus Y_i$$

Ako se i levoj i desnoj strani doda A_i, dobija se sledeći oblik prethodne jednačine:

$$A_i \oplus A_i \cdot B_i = A_i \oplus A_i \oplus Y_i$$
$$A_i \oplus A_i \cdot B_i = (A_i \oplus A_i) \oplus Y_i$$

imajući u vidu sledeće osobine Bulovih funkcija:

$$(A_i \oplus A_i) = A_i \cdot \overline{A_i} + \overline{A_i} \cdot A_i = 0$$
$$0 \oplus Yi = 0 \cdot \overline{Yi} + \overline{0} \cdot Yi = 1 \cdot Yi = Yi$$

izraz sa leve strane jednakosti se svodi na sledeći oblik:

$$Ai \oplus Ai \cdot \overline{Bi} = Yi$$
$$Y_i = A_i \cdot \overline{A_i \cdot B_i} + \overline{A_i} \cdot A_i \cdot B_i$$
$$Y_i = A_i \cdot \left(\overline{A_i} + \overline{B_i}\right) + 0 \cdot B_i$$
$$Y_i = A_i \cdot \overline{A_i} + A_i \cdot \overline{B_i} + 0$$
$$Y_i = 0 + A_i \cdot \overline{B_i}$$
$$Y_i = A_i \cdot \overline{B_i}$$

Za realizaciju logičke operacije ILI (DISJUNKCIJE) iskoristiće se pretpostavke navedene u prethodnom delu teksta. Iskoristiće se slična metodologija za određivanje vrednosti koja se dovodi na ulaz Y_i punog sabirača. Polazna jednačina je:

$$F_i = A_i + B_i = A_i \oplus Y_i$$

primenom iste metodologije dodavanja A_i i levoj i desnoj strani jednakosti dobija se sledeća jednačina:

$$Y_i = A_i \oplus (A_i + B_i)$$

odavde je:

$$Y_i = \overline{A_i} \cdot (A_i + B_i) + A_i \cdot \overline{(A_i + B_i)}$$
$$Y_i = \overline{A_i} \cdot A_i + \overline{A_i} \cdot B_i + A_i \cdot \overline{A_i} \cdot \overline{B_i}$$
$$Y_i = 0 + \overline{A_i} \cdot B_i + 0 \cdot \overline{B_i}$$
$$Y_i = \overline{A_i} \cdot B_i$$

Uvažavajući pretohodno izvedene jednačine formira se Tabela 8.2 u kojoj su dati odgovarajući ulazi za odgovarajuće operacije. Pri tome treba naglasti da se razlikuju C_{in0} i ostali C_{ini}. C_{in0} se neposredno formira u zavisnosti od odabrane operacije.

Operacija	X_i	Y_i	$C_{in\ i}$	$C_{in\ 0}$
A + B	A_i	B_i	$C_{out\ i}$	0
A - B	A_i	$\overline{B_i}$	$C_{out\ i}$	1
$A \vee B$	A_i	$\overline{A_i} \cdot B_i$	0	0
$A \wedge B$	A_i	$A_i \cdot \overline{B_i}$	0	0

Tabela 8.2: Ulazi punog sabirača za pojedine operacije

Na osnovu prethodne tabele formiraju se odgovarajuće funkcije za ulaze punog sabirača:

$$X_i = A_i$$

$$Y_i = \overline{S_1} \cdot \overline{S_0} \cdot B_i + \overline{S_1} \cdot S_0 \cdot \overline{B_i} + S_1 \cdot \overline{S_0} \cdot \overline{A_i} \cdot B_i + S_1 \cdot S_0 \cdot A_i \cdot \overline{B_i}$$

$$C_{in\ i} = \overline{S_1} \cdot C_{out\ i-1}$$

$$C_0 = \overline{S_1} \cdot S_0$$

Po zahtevu zadatka treba izvršiti sintezu pomoću NI kola pa se ove funkcije primenom De Morganovog zakona transformišu u sledeći oblik:

$$X_i = A_i$$

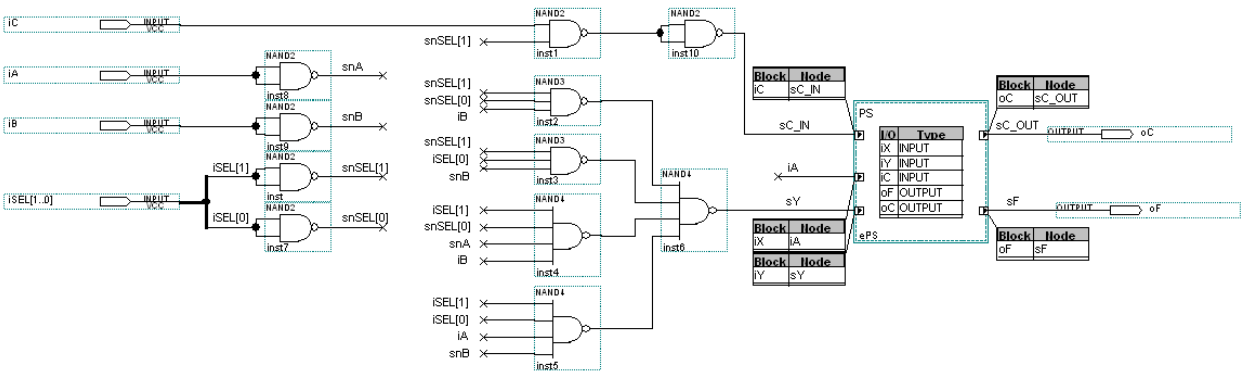
$$Y_i = \overline{\overline{S_1} \cdot \overline{S_0} \cdot B_i + \overline{S_1} \cdot S_0 \cdot \overline{B_i} + S_1 \cdot \overline{S_0} \cdot \overline{A_i} \cdot B_i + S_1 \cdot S_0 \cdot A_i \cdot \overline{B_i}}$$

$$Y_i = \overline{\overline{S_1} \cdot \overline{S_0} \cdot B_i} \cdot \overline{\overline{S_1} \cdot S_0 \cdot \overline{B_i}} \cdot \overline{S_1 \cdot \overline{S_0} \cdot \overline{A_i} \cdot B_i} \cdot \overline{S_1 \cdot S_0 \cdot A_i \cdot \overline{B_i}}$$

$$C_{in\ i} = \overline{\overline{S_1} \cdot C_{out\ i-1}}$$

$$C_0 = \overline{\overline{S_1} \cdot S_0}$$

Imajući u vidu gornje jednačine, realizaciju jednog stepena ALJ prikazuje Slika 8.8, pri tome treba imati u vidu da je, što se tiče prenosa, prvi stepen (LSB) različit u odnosu na ostale. Četvorobitna ALJ se sintetizuje ulančavanjem više jednobitnih stepena, Slika 8.10.



Slika 8.8: Realizacija jednog stepena ALJ sa NI kolima

U drugom delu javlja se problem sinteze pomerača. Kao i prethodnom delu i sinteza pomerača će se izvesti pomoću NI kola. Tabela 8.3 prikazuje prenosnu funkciju pomerača.

Operacija	S_3	S_2	H_3	H_2	H_1	H_0
ZERO	0	0	0	0	0	0
shl F	0	1	F_2	F_1	F_0	0
shr F	1	0	0	F_3	F_2	F_1
propuštanje	1	1	F_3	F_2	F_1	F_0

Tabela 8.3: Prenosna funkcija pomerača

Na osnovu prethodne tabele formiraju se prenosne funkcije sva četiri izlaza (H_3, H_2, H_1 i H_0) koje glase:

$$H_3 = 0 \cdot \overline{S_3} \cdot \overline{S_2} + F_2 \cdot \overline{S_3} \cdot S_2 + 0 \cdot S_3 \cdot \overline{S_2} + F_3 \cdot S_3 \cdot S_2$$
$$H_2 = 0 \cdot \overline{S_3} \cdot \overline{S_2} + F_1 \cdot \overline{S_3} \cdot S_2 + F_3 \cdot S_3 \cdot \overline{S_2} + F_2 \cdot S_3 \cdot S_2$$
$$H_1 = 0 \cdot \overline{S_3} \cdot \overline{S_2} + F_0 \cdot \overline{S_3} \cdot S_2 + F_2 \cdot S_3 \cdot \overline{S_2} + F_1 \cdot S_3 \cdot S_2$$
$$H_0 = 0 \cdot \overline{S_3} \cdot \overline{S_2} + 0 \cdot \overline{S_3} \cdot S_2 + F_1 \cdot S_3 \cdot \overline{S_2} + F_0 \cdot S_3 \cdot S_2$$

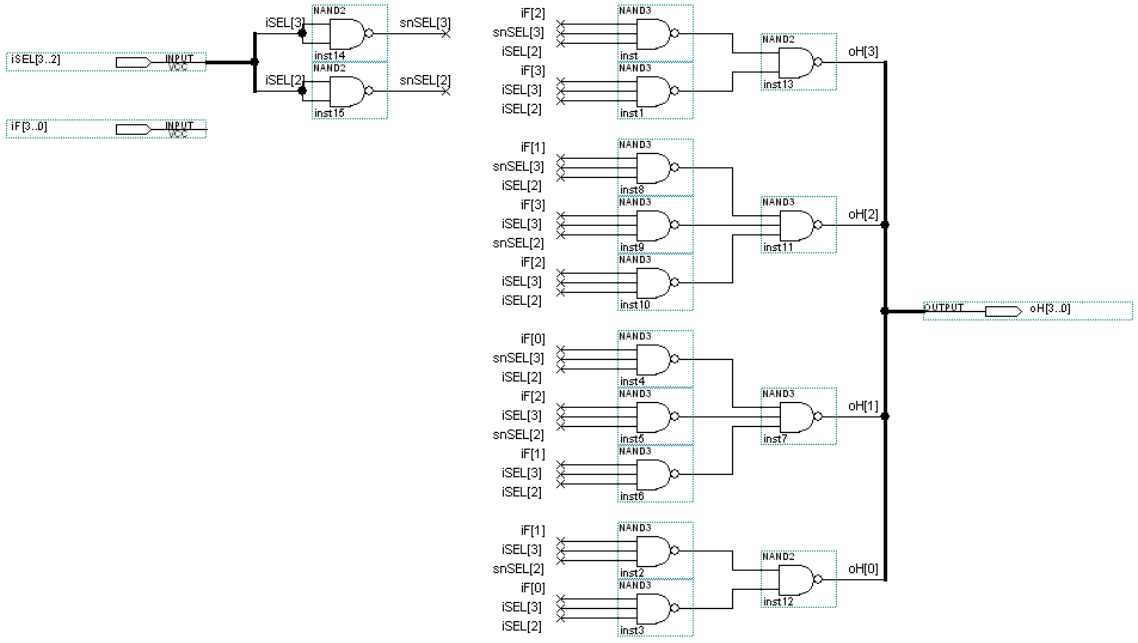
Ukoliko se eleminišu članovi uz koje figuriše nula, dobija se da u prenosnim funkcijama pomerača H_3 i H_0 figurišu samo dva prizvoda, a u funkcijama H_2 i H_1 po tri proizvoda:

$$H_3 = F_2 \cdot \overline{S_3} \cdot S_2 + F_3 \cdot S_3 \cdot S_2$$
$$H_2 = F_1 \cdot \overline{S_3} \cdot S_2 + F_3 \cdot S_3 \cdot \overline{S_2} + F_2 \cdot S_3 \cdot S_2$$
$$H_1 = F_0 \cdot \overline{S_3} \cdot S_2 + F_2 \cdot S_3 \cdot \overline{S_2} + F_1 \cdot S_3 \cdot S_2$$
$$H_0 = F_1 \cdot S_3 \cdot \overline{S_2} + F_0 \cdot S_3 \cdot S_2$$

Prevođenje jednačina u formu pogodnu za implementaciju pomoću NI kola se realizuje primenom I De Morganovog zakona na gornji set jednačina:

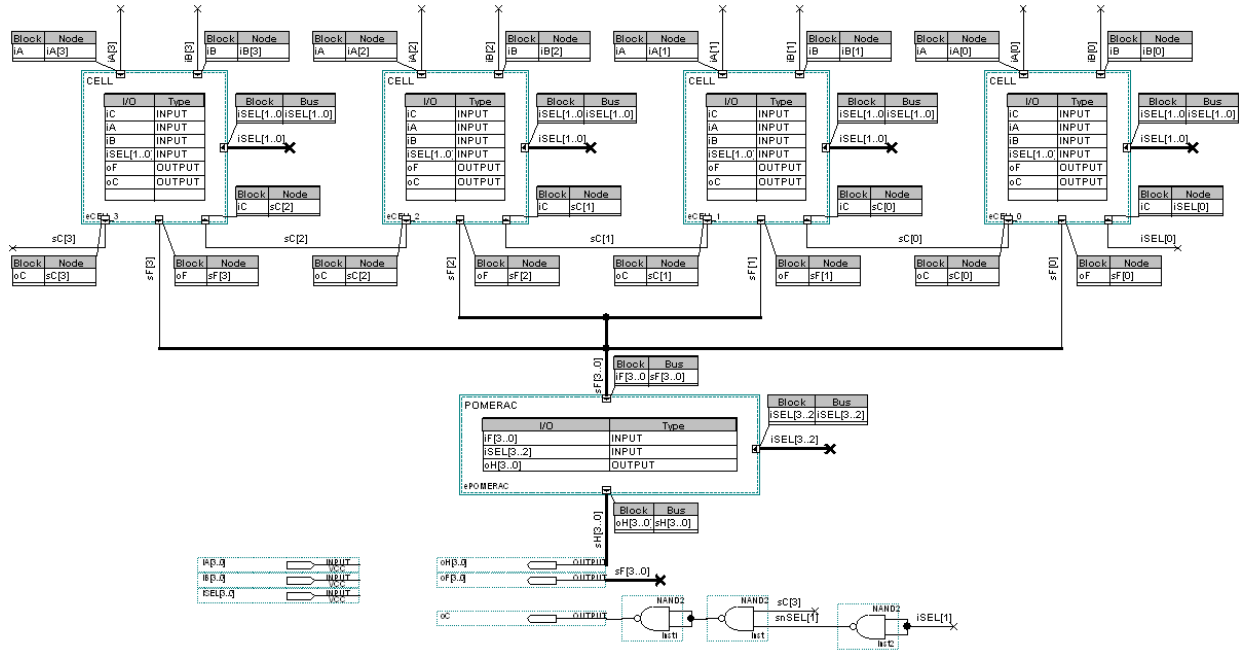
$$H_3 = \overline{\overline{F_2 \cdot \overline{S_3} \cdot S_2} \cdot \overline{F_3 \cdot S_3 \cdot S_2}}$$
$$H_2 = \overline{\overline{F_1 \cdot \overline{S_3} \cdot S_2} \cdot \overline{F_3 \cdot S_3 \cdot \overline{S_2}} \cdot \overline{F_2 \cdot S_3 \cdot S_2}}$$
$$H_1 = \overline{\overline{F_0 \cdot \overline{S_3} \cdot S_2} \cdot \overline{F_2 \cdot S_3 \cdot \overline{S_2}} \cdot \overline{F_1 \cdot S_3 \cdot S_2}}$$
$$H_0 = \overline{\overline{F_1 \cdot S_3 \cdot \overline{S_2}} \cdot \overline{F_0 \cdot S_3 \cdot S_2}}$$

Slika 8.9 prikazuje realizaciju pomerača pomoću NI kola.



Slika 8.9: Realizacija pomerača pomoću NI kola

Logičku šemu realizacije tražene aritmetičko logičke jedinice u celosti prikazuje Slika 8.10. Realizacija je izvršena modularno, tako da svaki modul ima svoje mesto u hijerarhi ALJ u skladu sa postavkom zadatka, Slika 8.6.



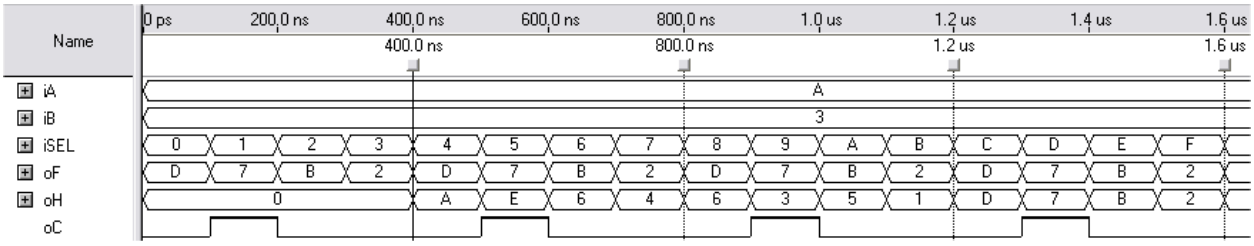
Slika 8.10: Logička šema realizovane ALJ

Slika 8.10 prikazuje logičku šemu realizovane ALJ, sa tom razlikom da je signal izlaznog prenosa (oC) realizovan na drugačiji način. Naime, pošto kod izvršenja logičkih operacija izlazni prenos nema nikakvu ulogu, u ovoj realizaciji je taj signal forsiran na 0 za logičke operacije. Pošto se logičke operacije

izvršavaju kada je `iSEL[1]=1`, a aritmetičke kada je `iSEL[1]=0`, izlazni prenos sa poslednjeg stepena ALJ se propušta na izlaz samo za aritmetičke operacije, tj. kada je `iSEL[1]=0`. Odgovarajuća jednačina je prema tome:

$$oC = sC[3] \cdot \overline{iSEL[1]} = \overline{\overline{sC[3] \cdot \overline{iSEL[1]}}}$$

Slika 8.11 prikazuje simulaciju rada realizovane ALJ. Prikazani su sve moguće operacije na jednom paru ulaznih operanada ($A=A_{hex}$; $B=3_{hex}$).



Slika 8.11: Vremenski dijagram simulacije rada realizovane ALJ

Vremenski dijagram je podeljen na 4 dela, na osnovu vrednosti selekcionih promenljivih `iSEL[3..2]`. Vektor `oF` predstavlja međurezultat, odnosno izlaz iz aritmetičkog dela ALJ i ulaz u pomerač. Njegov rezultat zavisi od vrednosti selekcionih promenljivih `iSEL[1..0]`, pa se zbog toga vrednost ovog vektora ponavlja na svakih 400ns.

U prvom je delu, do 400 ns, `iSEL[3..2]=00`$_{bin}$, pa se na osnovu postavke zadatka na izlazu ALJ (vektor `oH`) daje vrednost nula. Drugi deo, od 400ns do 800ns, selecione promenljive imaju vrednost `iSEL[3..2]=01`$_{bin}$ što na izlazu ALJ daje rezultat `oF` pomeren ulevo za jedno mesto. Pomeranje udesno za jedno mesto se izvršava u trežem delu, od 800ns do 1200ns, jer u tom intervalu važi da je `iSEL[3..2]=10`$_{bin}$. Poslednji interval, od 1200ns do 1600ns, na izlaz ALJ propušta neizmenjen rezultat `oF`.

8.2 ZADATAK:

ALJ iz prethodnog zadatka realizovati pomoću VHDL jezika za opis fizičke arhitekture.

a) ALJ realizovati modularno na osnovu relacija koje su izvedene u rešenju prethodnog zadatka.
b) ALJ realizovati pomoću vektora

REŠENJE:

a)

Na osnovu jednačina koje su izvedene u prethodnom zadatku, ALJ se može modularno realizovati i uz pomoć VHDL jezika za opis fizičke arhitekture.

Kao i u prethodnom zadatku, prvi korak je realizacija punog sabirača. VHDL kod prikazan u nastavku teksta je realizovan na osnovu poznatih prenosnih funkcija punog sabirača.

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY PUN_SABIRAC IS PORT (
  iX, iY, iC: IN  std_logic;
  oS, oC:     OUT std_logic );
END PUN_SABIRAC;

ARCHITECTURE ARH_PUN_SABIRAC OF PUN_SABIRAC IS BEGIN
  oS <= (iX XOR iY XOR iC);
  oC <= (iX AND iY) OR (iX AND iC) OR (iY AND iC);
END ARH_PUN_SABIRAC;
```

Nakon realizacije punog sabirača, sledi realizacija jednobitne ćelije ALJ. Jednobitna ćelija se bazira na punom sabiraču i odgovarajuće kombinacione mreže koja u zavisnosti od selektujućih signala na ulaze punog sabirača dovodi odgovarajuće vrednosti za realizaciju odabrane operacije. U prethodnom zadatku su izvedene sledeće relacije za implementaciju kombinacione mreže:

$$X_i = A_i$$

$$Y_i = \overline{S_1} \cdot \overline{S_0} \cdot B_i + \overline{S_1} \cdot S_0 \cdot \overline{B_i} + S_1 \cdot \overline{S_0} \cdot \overline{A_i} \cdot B_i + S_1 \cdot S_0 \cdot A_i \cdot \overline{B_i}$$

$$C_{in\,i} = \overline{S_1} \cdot C_{out\,i-1}$$

$$C_0 = \overline{S_1} \cdot S_0$$

Na osnovu navedenih relacija sledi VHDL implementacija jednobitne ćelije tražene ALJ.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY CELL IS PORT(
  iA, iB, iC: IN  std_logic;
  iSEL:       IN  std_logic_vector(1 DOWNTO 0);
  oS, oC:     OUT std_logic);
END CELL;

ARCHITECTURE ARH_CELL OF CELL IS

  -- opis sprege komponente koja se
  -- instancira u okviru arhitekture
  COMPONENT PUN_SABIRAC PORT (
    iX, iY, iC: IN  std_logic;
    oS, oC:     OUT std_logic );
  END COMPONENT;

 -- signali koji formiraju ulaze u pun sabirac
 SIGNAL sX, sY, sC: std_logic;
```

```
BEGIN
  sX <= iA;
  sY <= ((NOT(iSEL(1)) AND NOT(iSEL(0))) AND                  iB)  OR
        ((NOT(iSEL(1)) AND     iSEL(0) ) AND              NOT(iB)) OR
        ((    iSEL(1)  AND NOT(iSEL(0))) AND (NOT(iA) AND     iB)) OR
        ((    iSEL(1)  AND     iSEL(0) ) AND (    iA  AND NOT(iB)));
  sC <= NOT(iSEL(1)) AND iC;

  ePUN_SABIRAC: PUN_SABIRAC PORT MAP(iX=>sX, iY=>sY, iC=>sC,
                                     oS=>oS, oC=>oC);
END ARH_CELL;
```

Pošto je za realizaciju ćelije ALJ potreban pun sabirač, on se mora instancirati u VHDL kodu. Da bi VHDL prevodilac saznao za modul (entitet, komponentu) koja se želi instancirati, mora se navesti deklaracija sprege date komponente pre same instance. U ovom slučaju to je urađeno u zaglavlju arhitekture VHDL opisa jednobitne ćelije tražene ALJ.

Realizacijom ćelije ALJ, stvoreni su uslovi za realizaciju dela ALJ zaduženog za izvršenje aritmetičkih i logičkih operacija (ulančavanjem više jednobitnih stepena). U nastavku sledi implementacija traženog pomerača.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY POMERAC IS PORT(
  iH:   IN  std_logic_vector(3 DOWNTO 0);
  iSEL: IN  std_logic_vector(3 DOWNTO 2);
  oH:   OUT std_logic_vector(3 DOWNTO 0) );
END POMERAC;

ARCHITECTURE ARH_POMERAC OF POMERAC IS
BEGIN
  PROCESS (iSEL, iH) BEGIN
    CASE iSEL IS
                          -- ZERO
      WHEN "00"   => oH <= "0000";
                          -- u levo za 1 mesto
      WHEN "01"   => oH <= (iH(2 DOWNTO 0) & '0');
                          -- u desno za 1 mesto
      WHEN "10"   => oH <= ('0' & iH(3 DOWNTO 1));
                          -- propustanje
      WHEN OTHERS => oH <= iH;
    END CASE;
  END PROCESS;
END ARH_POMERAC;
```

Pomerač u ovom slučaju nije realizovan pomoću izvedenih relacija, već agregacijom signala. Time je postignuta veća čitljivost VHDL koda.

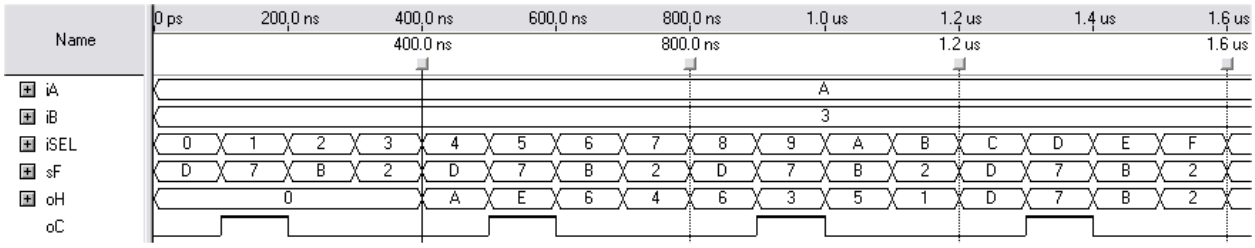
Sledi VHDL kod implementacije kompletne četvorobitne ALJ, gde je ulančano 4 jednobitnih ćelija i pomerač. Kao i kod rešenja prethodnog zadatka i ovde je vrednost signala izlaznog prenosa ograničena samo za aritmetičke operacije, inače ovaj signal ima vrednost nula.

```
-----------------------------------------------------------------
--  Pakovanje sa opisom komponenti koje ce biti instancirane
--  u entitetu koji se realizuje:
--     1. CELL    - jednobitna celija ALJ
--     2. POMERAC - pomerac
-----------------------------------------------------------------
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

PACKAGE ALJ01_PKG IS
  COMPONENT CELL PORT (
    iA, iB, iC: IN  std_logic;
    iSEL:       IN  std_logic_vector(1 DOWNTO 0);
    oS, oC:     OUT std_logic);
  END COMPONENT;

  COMPONENT POMERAC  PORT(
    iH:   IN  std_logic_vector(3 DOWNTO 0);
    iSEL: IN  std_logic_vector(3 DOWNTO 2);
    oH:   OUT std_logic_vector(3 DOWNTO 0) );
  END COMPONENT;
END ALJ01_PKG;

LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE work.ALJ01_PKG.all; -- ukljucivanje pakovanja sa komponentama

ENTITY ALJ01 IS PORT(
  iA, iB: IN  std_logic_vector(3 DOWNTO 0);
  iSEL:   IN  std_logic_vector(3 DOWNTO 0);
  oH:     OUT std_logic_vector(3 DOWNTO 0);
  oC:     OUT std_logic );
END ALJ01;

ARCHITECTURE ARH_ALJ01 OF ALJ01 IS
  -- vektori za povezivanje modula ALJ
             -- izlazni prenosi iz celija
  SIGNAL sC: std_logic_vector(3 DOWNTO 0);
             -- rezultat operacije, ulaz u pomerac
  SIGNAL sF: std_logic_vector(3 DOWNTO 0);
BEGIN

  eCELL_0:CELL PORT MAP (iA=>iA(0),   iB=>iB(0),
                         iC=>iSEL(0), iSEL=>iSEL(1 DOWNTO 0),
                         oS=>sF(0),   oC=>sC(0));
  eCELL_1:CELL PORT MAP (iA=>iA(1),   iB=>iB(1),
                         iC=>sC(0),   iSEL=>iSEL(1 DOWNTO 0),
                         oS=>sF(1),   oC=>sC(1));
  eCELL_2:CELL PORT MAP (iA=>iA(2),   iB=>iB(2),
                         iC=>sC(1),   iSEL=>iSEL(1 DOWNTO 0),
                         oS=>sF(2),   oC=>sC(2));
  eCELL_3:CELL PORT MAP (iA=>iA(3),   iB=>iB(3),
                         iC=>sC(2),   iSEL=>iSEL(1 DOWNTO 0),
                         oS=>sF(3),   oC=>sC(3));

  ePOMERAC: POMERAC
            PORT MAP (iH=>sF, iSEL=>iSEL(3 DOWNTO 2), oH=>oH);
```

```
    -- generisanje izlaznog prenosa samo za aritmeticke operacije
    oC <= sC(3) AND NOT(iSEL(1));

END ARH_ALJ01;
```

U okviru arhitekture `ARH_ALJ01` se instanciraju jednobitne ćelije i pomerač. Deklaracija sprega ovih komponenti se može navesti u zaglavlju arhitekture, slično kao što je urađeno kod implementacije jednobitne ćelije. Ovde je formirano posebno pakovanje, `ALJ01_PKG`, koje sadrži deklaracije sprega entiteta `CELL` i `POMERAC`. Nakon toga je ovo pakovanje uključeno u proces prevođenja iskazom `USE work.ALJ01_PKG.all`.

Slika 8.12 prikazuje simulaciju rada realizovane ALJ. Kao što se i očekuje, funkcionalnost ALJ je potpuno ekvivalentna sa rešenjem prethodnog zadatka, tj. za iste ulazne signale generišu se iste izlazne vrednosti.



Slika 8.12: Simulacija rada realizovane ALJ na nivou modula

b)

Za realizaciju ALJ na nivou vektora nije potreban nikakav proces minimizacije Bulovih jednačina. Prilikom ovakvog rešavanja se pomoću VHDL jezika za opis fizičke arhitekture opiše funkcionalnost ALJ, a proces minimizacije i određivanja Bulovih relacija za implementaciju iste se prepušta VHDL prevodiocu. Ovde je prikzan jedan od načina opisa tražene ALJ.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;

ENTITY ALJ01 IS PORT(
  iA, iB: IN  unsigned(3 DOWNTO 0);
  iSEL:   IN  std_logic_vector(3 DOWNTO 0);
  oH:     OUT unsigned(3 DOWNTO 0);
  oC:     OUT std_logic );
END ALJ01;

ARCHITECTURE ARH_ALJ01 OF ALJ01 IS
  -- rezultat operacije, ulaz u pomerac
  -- prosiren za jedan bit gde ce se smestiti
  -- izracunati izlazni prenos
  SIGNAL sF: unsigned(4 DOWNTO 0);
BEGIN
```

```
  -- aritmeticko-logicki deo ALJ
  PROCESS(iSEL, iA, iB)
   -- operandi koji su prosireni za jedan bit
   -- zbog racunanja izlaznog renosa
   VARIABLE vA, vB: unsigned(4 DOWNTO 0);
  BEGIN
   vA := ('0' & iA); -- operand A prosiren za jedan bit
   vB := ('0' & iB); -- operand B prosiren za jedan bit

   CASE iSEL(1 DOWNTO 0) IS
    WHEN "00"   => sF <= vA + vB;                    -- sabiranje
    WHEN "01"   => sF <= vA + (('0' & NOT(iB))) + 1; -- oduzimanje
    WHEN "10"   => sF <= ('0' & (iA OR  iB));        -- logicko ILI
    WHEN OTHERS => sF <= ('0' & (iA AND iB));        -- logicko I
   END CASE;
  END PROCESS;

  -- pomerac
  PROCESS (iSEL, sF) BEGIN
   CASE iSEL(3 DOWNTO 2) IS
    WHEN "00"   => oH <= "0000";                 -- ZERO
    WHEN "01"   => oH <= (sF(2 DOWNTO 0) & '0'); -- ulevo za 1 mesto
    WHEN "10"   => oH <= ('0' & sF(3 DOWNTO 1)); -- udesno za 1 mesto
    WHEN OTHERS => oH <= sF(3 DOWNTO 0);         -- propustanje
   END CASE;
  END PROCESS;

  -- signal izlaznog prenosa,
  -- ogranicen samo za aritmeticke operacije
  oC <= std_logic(sF(4)) AND NOT(iSEL(1));
 END ARH_ALJ01;
```

Prva razlika u odnosu na prethodnu implementaciju se uočava u opisu entiteta, jer su u ovom rešenju promenjeni tipovi ulaznih operanada i izlaznog vektora ALJ. Promena je uvedena zbog toga što VHDL standard ne definiše aritmetičke operacije nad vektorima tipa `std_logic_vector`. Aritmetičke operacije su definisane nad tipovima `unsigned`, `signed` i `integer`. Definicija ovih operacija se nalazi u okviru pakovanja `numeric_std` koje je sastavni deo biblioteke `ieee`. Zbog toga je ovo pakovanje uključeno u proces prevođenja iskazom `USE ieee.numeric_std.all`.

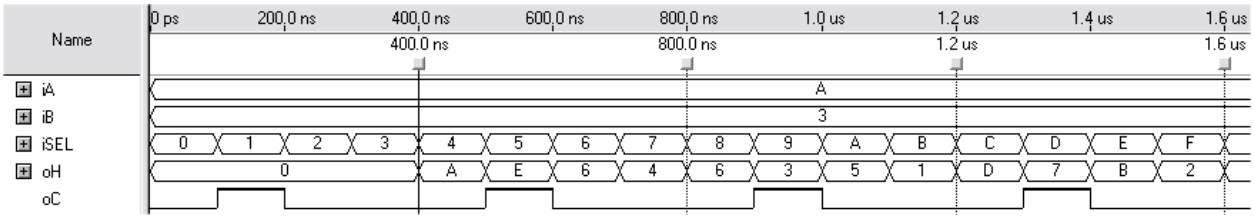
Arhitektura ovog rešenja se sastoji od dva procesa. U jednom procesu su implementirane aritmetičke i logičke operacije, dok drugi proces opisuje pomerač.

Za realizaciju aritmetičkih operacija javlja se problem generisanja izlaznog prenosa. U ovom rešenju je to postignuto tako da su ulazni operandi prošireni sa gornje strane sa dodatnim bitom čija je vrednost ‘0’. Ovde su za tu potrebu iskorišćene promenljive, variable, `vA` i `vB`. Sve aritmetičke operacije se izvršavaju nad tako formiranim promenljivama, a ne na ulaznim operandima. Nakon izvršenja aritmetičeke operacije, tj. sabiranja, vrednost izlaznog prenosa (`oC`) će biti smeštena na poziciji bita najveće važnosti rezultujućeg vektora (`sF`), a preostali biti istog vektora će sadržati rezultat operacije. U prikazanom rešenju oduzimanje je realizovano kao sabiranje sa drugim komplementom:

$$A - B = A + \overline{B} + 1$$

Realizacija pomerača je ista kao u prethodnom rešenju. Izlaz iz procesa koji opisuje aritmetičke i logičke operacije (`sF`) se pomera u zavisnosti od vrednosti odgovarajućih selektujućih signala (`iSEL(3 DOWTO 2)`).

Funkcionalnost ovakve realizacije je potpuno ekvivalentna sa prethodnim realizacijama, kao što i prikazuje Slika 8.13.



Slika 8.13: Simulacija rada realizovane ALJ na nivou vektora

8.3 ZADATAK:

Upotrebom standardnih metoda minimizacije isprojektovati ALJ računarskog sistema za obavljanje elementarnih operacija koje prikazuje Tabela 8.4.

Operacija	Selekcione promenljive		
	S_2	S_1	S_0
Y= A + B	0	0	0
Y= A - B	0	0	1
$Y=\overline{A}$	0	1	0
Y= A +1	0	1	1
ZERO	1	0	0
Y= shr A	1	0	1
Y= shl A	1	1	0
ZERO	1	1	1

Tabela 8.4: Operacije koje izvršava ALJ

ALJ realizovati pomoću VHDL jezika za opis fizičke arhitekture (modularna implementacija). Takođe, isprojektovati logičke šeme ekvivalentne sa dobivenim VHDL kodom. Pri tome za implementaciju pomerača iskoristiti multipleksere 4×1 sa oznakom 74253.

REŠENJE:

Sinteza ALJ počinje definisanjem prenosnih funkcija punog sabirača. Poznavanjem funkcije punog sabirača, kreiranjem tablice ulaza/izlaza punog sabirača i minimizacijom izlaznih funkcija dobiju se sledeće jednačine:

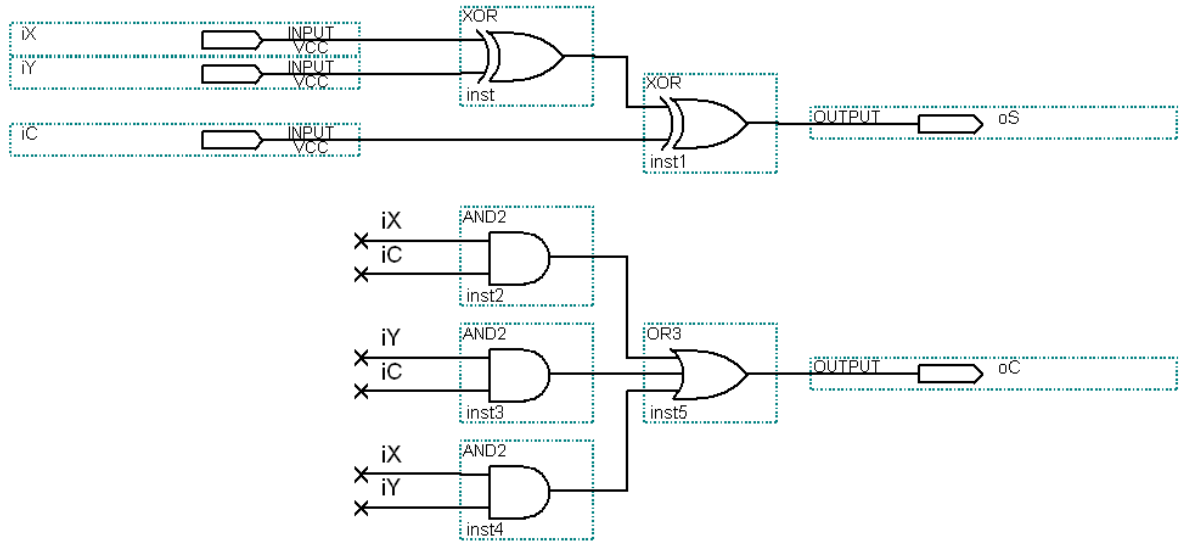
$$F = X_i \oplus Y_i \oplus C_{in}$$
$$C_{out} = X_i \cdot C_{in} + Y_i \cdot C_{in} + X_i \cdot Y_i$$

gde su ulazi punog sabirača označeni sa X_i, Y_i, ulazni prenos je označen sa C_{in}, dok je sa F označen izlaz iz punog sabirača, a izlazni prenos punog sabirača označen sa C_{out}. Na osnovu ovih jednačina formira se sledeći VHDL opis punog sabirača.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY PUN_SABIRAC IS PORT (
  iX, iY, iC: IN  std_logic;
  oS, oC:     OUT std_logic );
END PUN_SABIRAC;

ARCHITECTURE ARH_PUN_SABIRAC OF PUN_SABIRAC IS BEGIN
  oS <= (iX XOR iY XOR iC);
  oC <= (iX AND iY) OR (iX AND iC) OR (iY AND iC);
END ARH_PUN_SABIRAC;
```

Za ulaze punog sabirača su korišćene oznake `iX` i `iY`, dok je ulazni prenos predstavljen sa `iC`. Izlaz punog sabirača je označen sa `oS`, a izlazni prenos sa `oC`.

Logički šemu ekvivalentnu sa prethodnim VHDL opisom punog sabirača prikazuje Slika 8.14.



Slika 8.14: Logička šema punog sabirača

Da bi se realizovala ALJ potrebno je između ulaza punog sabirača i spoljnih ulaza same ALJ formirati odgovarajuću kombinacionu mrežu kojom se upravlja pomoću selekcionih promenljivih, a u cilju realizacije željene funkcije. Dakle, potrebno je, u sledećem koraku, formirati tabelu iz koje će se videti šta treba dovesti na ulaze punog sabirača kako bi se realizovala željena funkcija. Pre samog

popunjavanja tabele, treba izvršiti analizu pojedinačnih slučajeva za svaku od zadatih funkcija tražene ALJ.

$Y = A + B$ → Na ulaz X_i se dovodi A_i, dok se na ulaz Y_i se dovodi B_i;

$Y = A - B$ → Operaciju oduzimanja se može realizovati i kao sabiranje sa drugim komplementom, odnosno $A-B=A+\overline{B}+1$. Na osnovu toga će se na ulaz X_i dovesti A_i, na ulaz Y_i promenljiva B_i, a ulazni prenos u najniži razred biće $C_{in0}=1$;

$Y = \overline{A}$ → Na ulaz X_i se dovodi komplementirana vrednost promenljive A_i, $Y_i=0$;

$Y = A+1$ → Na ulaz X_i se dovodi A_i, s tim da će ulazni prenos u najniži razred biti $C_{in0}=1$;

$Y = shr\ A$ → Na ulaz X_i se dovodi A_i, dok na ulaz Y_i se dovodi 0. Time se izvrši propuštanje operanda A kroz aritmetičko-logički deo ALJ, dok će se samo pomeranje izvršiti u posebnom pomeračkom bloku.

$Y = shl\ A$ → Isto kao i prethodno.

Selekcija pojedinih operacija ALJ se vrši pomoću selekcionih promenljivih S_2, S_1 i S_0. Sada se može formirati tabela (Tabela 8.5).

Operacija	Ulazi PS			Selekcione promenljive		
	X_i	Y_i	C_{in0}	S_2	S_1	S_0
Y= A plus B	A_i	B_i	0	0	0	0
Y= A minus B	A_i	$\overline{B_i}$	1	0	0	1
$Y=\overline{A}$	$\overline{A_i}$	0	0	0	1	0
Y= A +1	A_i	0	1	0	1	1
ZERO	×	×	×	1	0	0
Y= shr A	A_i	0	0	1	0	1
Y= shl A	A_i	0	0	1	1	0
ZERO	×	×	×	1	1	1

Tabela 8.5: Ulazi punog sabirača i vrednosti selekcionih promenljivih za pojedine operacije

Da bi se realizovala tražena kombinaciona mreža, mora se najpre izvršiti minimizacija funkcija $X_i(A_i, B_i, S_2, S_1, S_0)$, $Y_i(A_i, B_i, S_2, S_1, S_0)$, $C_{in\ 0}(Ai, Bi, S_2, S_1, S_0)$ kao i $C_{in\ i}(S_2, S_1, S_0, C_{outi-1})$. Radi toga se najpre formiraju odgovarajuće tabele koje će se iskoristiti za formiranje Karnoovih karti, Tabela 8.6.

Na osnovu formiranih tabela mogu se formirati odgovarajuće Karnoove karte za minimizaciju Bulovih funkcija koje reprezentuju ulaze u pun sabirač, Slika 8.15.

Selekcione promenljive			A_i	X_i
S_2	S_1	S_0		
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	×
1	0	0	1	×
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	×
1	1	1	1	×

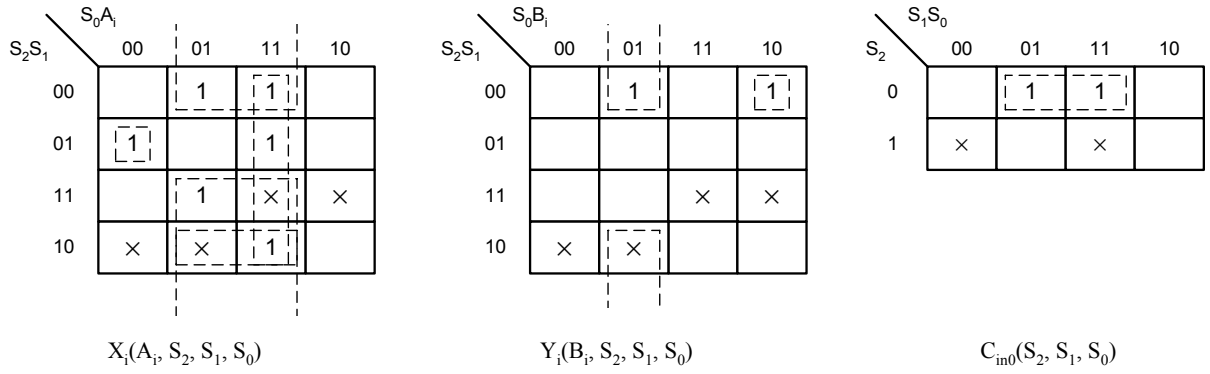
Selekcione promenljive			B_i	Y_i
S_2	S_1	S_0		
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	×
1	0	0	1	×
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	×
1	1	1	1	×

Selekcione promenljive			C_{in0}	C_{ini}
S_2	S_1	S_0		
0	0	0	0	C_{outi-1}
0	0	1	1	C_{outi-1}
0	1	0	0	C_{outi-1}
0	1	1	1	C_{outi-1}
1	0	0	×	×
1	0	1	0	×
1	1	0	0	×
1	1	1	×	×

Tabela 8.6: Tabele za minimizaciju ulaznih funkcija u pun sabirač

Direktno iz tabele se vidi da će funkcija za C_{ini} biti oblika:

$$C_{ini} = \overline{S_2} \cdot C_{outi-1}$$



Slika 8.15: Karnoove karte funkcija X_i, Y_i, C_{in0}

Minimizacijom uz pomoć Karnoovih mapa se dobijaju sledeće funkcije koje se dovode na ulaze odgovarajućih punih sabirača:

$$X_i = \overline{S_2} \cdot S_1 \cdot \overline{S_0} \cdot \overline{A_i} + S_2 \cdot A_i + \overline{S_1} \cdot A_i + S_0 \cdot A_i = \overline{S_2} \cdot S_1 \cdot \overline{S_0} \cdot \overline{A_i} + A_i \cdot (S_2 + \overline{S_1} + S_0)$$

$$Y_i = \overline{S_1} \cdot \overline{S_0} \cdot B_i + \overline{S_2} \cdot \overline{S_1} \cdot S_0 \cdot \overline{B_i} \cdot$$

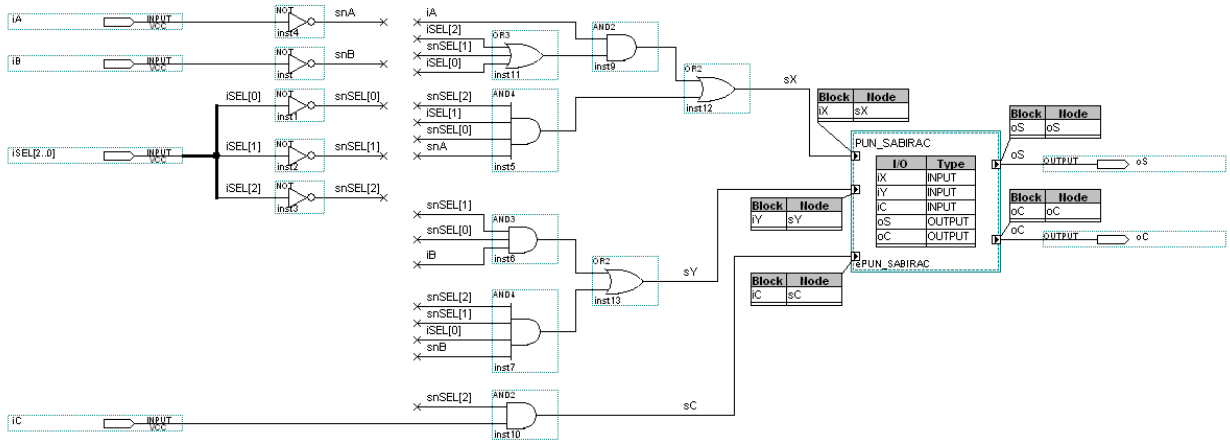
$$C_{in0} = \overline{S_2} \cdot S_0$$

Sada se može pristupiti formiranju VHDL opisa jednobitne ćelije aritmetičko-logičke. U VHDL kodu su sa `iA`, `iB` i `iC` označeni ulazni biti operanada A i B i ulazni prenos respektivno. `iSEL` obeležava vektor za odabir operacije. Rezultat i izlazni prenos su predstavljeni sa oznakama `oS` i `oC` redom.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY CELL IS PORT(
  iA, iB, iC: IN  std_logic;
  iSEL:       IN  std_logic_vector(2 DOWNTO 0);
  oS, oC:     OUT std_logic);
END CELL;

ARCHITECTURE ARH_CELL OF CELL IS
  -- opis sprege komponente koja se
  -- instancira u okviru arhitekture
  COMPONENT PUN_SABIRAC PORT (
    iX, iY, iC: IN  std_logic;
    oS, oC:     OUT std_logic );
  END COMPONENT;

  -- signali koji formiraju ulaze u pun sabirac
  SIGNAL sX, sY, sC: std_logic;
BEGIN

 sX <= (NOT(iSEL(2)) AND iSEL(1) AND NOT(iSEL(0)) AND NOT(iA)) OR
       ((iSEL(2) OR NOT(iSEL(1)) OR iSEL(0)) AND iA);
 sY <= (                  NOT(iSEL(1)) AND NOT(iSEL(0)) AND     iB) OR
       (NOT(iSEL(2)) AND NOT(iSEL(1)) AND     iSEL(0)  AND NOT(iB));
 sC <=  NOT(iSEL(2)) AND iC;
```

```
  ePUN_SABIRAC: PUN_SABIRAC
          PORT MAP(iX=>sX, iY=>sY, iC=>sC,
                   oS=>oS, oC=>oC);

END ARH_CELL;
```

Prethodni VHDL opis jednobitne ćelije aritmetičko logičke jedinice se može predstaviti logičkon šemom koju prikazuje Slika 8.16.



Slika 8.16: Logička šema jednobitne ćelije ALJ

Nakon isprojektovane jednobitne ćelije sledi projektovanje traženog pomerača. Tabela 8.7 prikazuje izlaze pomerača u funkciji od ulaznih selekcionih promenljivih i ulaza u pomerač.

Selekcione promenljive			Operacija	Izlazi pomerača			
S_2	S_1	S_0		H_3	H_2	H_1	H_0
0	×	×	propuštanje	F_3	F_2	F_1	F_0
1	0	0	nula na izlazu	0	0	0	0
1	0	1	shr F	0	F_3	F_2	F_1
1	1	0	shl F	F_2	F_1	F_0	0
1	1	1	nula na izlazu	0	0	0	0

Tabela 8.7: Funkcionalna tabela pomerača

Na osnovu prethodne tabele se lako formira VHDL opis traženog pomerača, koristeći odgovarajuću agregaciju signala u zavisnosti od vrednosti selekcionih promenljivih.

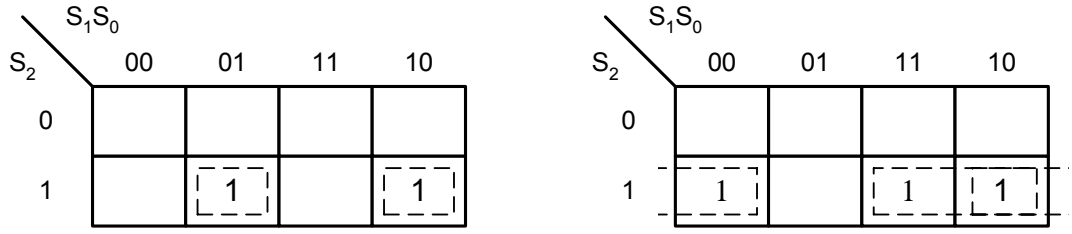
```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY POMERAC IS PORT(
  -- selekcione promenljive
  iSEL: IN  std_logic_vector(2 DOWNTO 0);
  -- ulazni vektor
  iH:   IN  std_logic_vector(3 DOWNTO 0);
  -- izlazni vektor
  oH:   OUT std_logic_vector(3 DOWNTO 0) );
END POMERAC;

ARCHITECTURE ARH_POMERAC OF POMERAC IS
BEGIN

  PROCESS (iSEL, iH) BEGIN
    CASE iSEL IS
      WHEN "100"  => oH <= "0000";
      WHEN "101"  => oH <= ('0' & iH(3 DOWNTO 1));
      WHEN "110"  => oH <= (iH(2 DOWNTO 0) & '0');
      WHEN "111"  => oH <= "0000";
      WHEN OTHERS => oH <= iH;
    END CASE;
  END PROCESS;

END ARH_POMERAC;
```

Fomiranje pomerača na osnovu multipleksera ne sledi direktno iz prethodnog VHDL koda. Prvo je potrebno odrediti prenosne funkcije za generisanje adresnih signala multipleksera. Ako te signale označimo sa D i C, tada Tabela 8.7 prikazuje izlaze pomerača u funkciji od ulaznih selekcionih promenljivih i ulaza u pomerač, kao i funkcije adresnih signala multipleksera u funkciji od ulaznih selekcionih promenljivih.

Selekcione promenljive			Operacija	Adresni signali		Izlazi pomerača			
S_2	S_1	S_0		D	C	H_3	H_2	H_1	H_0
0	×	×	propuštanje	0	0	F_3	F_2	F_1	F_0
1	0	0	nula na izlazu	0	1	0	0	0	0
1	0	1	shr F	1	0	0	F_3	F_2	F_1
1	1	0	shl F	1	1	F_2	F_1	F_0	0
1	1	1	nula na izlazu	0	1	0	0	0	0

Tabela 8.8: Funkcionalna tabela pomerača

Vrednosti adresnih signala multipleksera u pojedinim slučajevima su proizvoljno usvojene, s obzirom da nisu unapred zadate. Na osnovu prethodne table mogu se formirati Karnoove mape za minimizaciju funkcija $D(S_2, S_1, S_0)$ i $C(S_2, S_1, S_0)$, kao što je to prikazuje Slika 8.17.



$D(S_2, S_1, S_0)$ $C(S_2, S_1, S_0)$

Slika 8.17: Karnoove karte funkcija D (S_2,S_1,S_0) i C(S_2,S_1,S_0

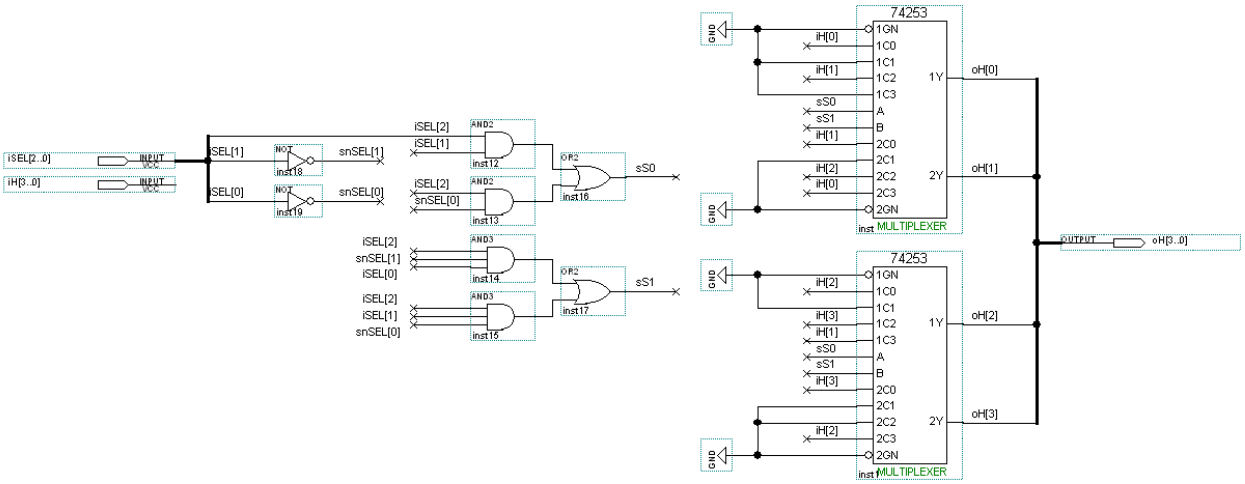
Funkcije D i C će nakon minimizacije biti sledećeg oblika:

$$D = S_2 \cdot \overline{S_1} \cdot S_0 + S_2 \cdot S_1 \cdot \overline{S_0}$$

$$C = S_2 \cdot S_1 + S_2 \cdot \overline{S_0}$$

Sada se može pristupiti formiranju logičke šeme pomerača, Slika 8.18 Adresni isgnali C i D su na slici označeni sa `sS0` i `sS1` respektivno.

Pošto su formirani svi moduli ALJ, pristupa se povezivanju istih sa ciljem realizacije tražene četvorobitne ALJ. Aritmetičko-logički deo ALJ se formira ulančavanjem četiri jednobitne ćelije, vodeći računa o ulaznom prenosu prvog stepena. Tako formirani rezultujući vektor se dovodi na ulaz pomerača, dok izlaz pomerača predstavlja krajnji rezultat realizovane ALJ.

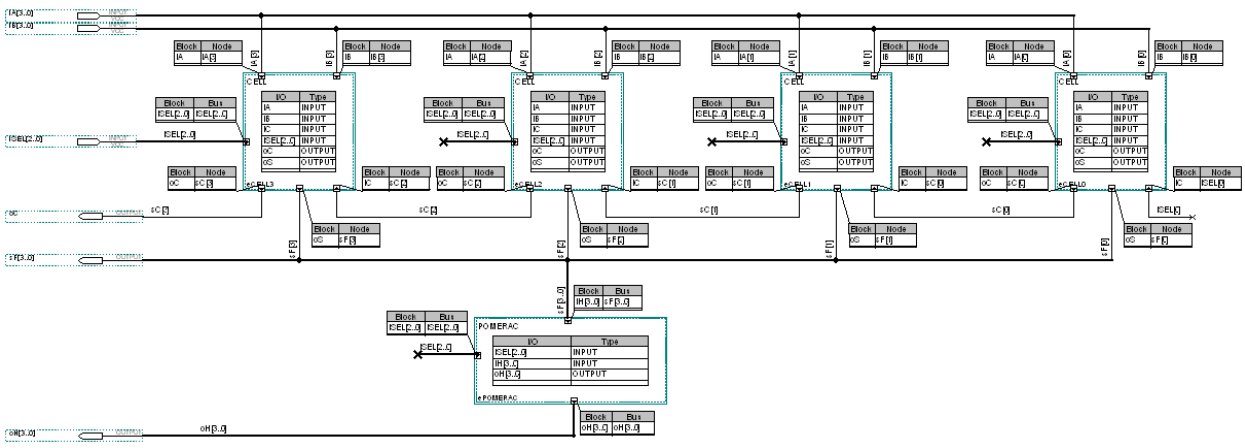


Slika 8.18: Logička šema realizovanog pomerača

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY ALJ02 IS PORT(
  iA, iB: IN  std_logic_vector(3 DOWNTO 0);   -- ulazni parametri
  iSEL:   IN  std_logic_vector(2 DOWNTO 0);   -- selekcija operacije
  oH:     OUT std_logic_vector(3 DOWNTO 0);   -- rezultat
  oC:     OUT std_logic);                     -- izlazni prenos
END ALJ02;
```

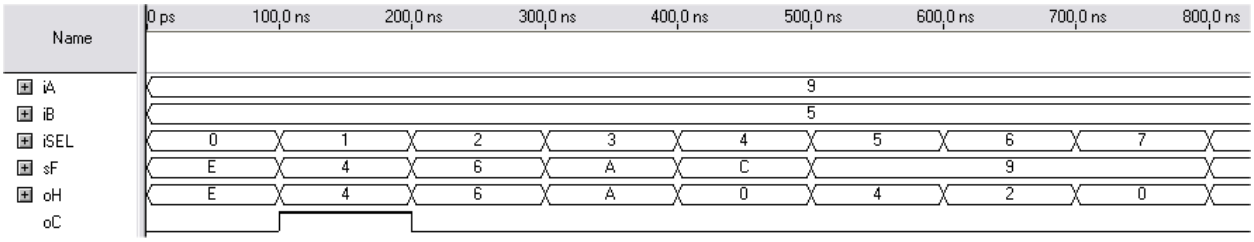
```
ARCHITECTURE ARH_ALJ02 OF ALJ02 IS
  -- opis sprege komponenti koje se
  -- instanciraju u okviru arhitekture
  COMPONENT CELL PORT (
    iA, iB, iC: IN  std_logic;
    iSEL:       IN  std_logic_vector(2 DOWNTO 0);
    oS, oC:     OUT std_logic);
  END COMPONENT;

  COMPONENT POMERAC  PORT(
    iH:   IN  std_logic_vector(3 DOWNTO 0);
    iSEL: IN  std_logic_vector(2 DOWNTO 0);
    oH:   OUT std_logic_vector(3 DOWNTO 0) );
  END COMPONENT;

  -- vektori za povezivanje modula ALJ
            -- izlazni prenosi iz celija
  SIGNAL sC: std_logic_vector(3 DOWNTO 0);
            -- rezultat operacije, ulaz u pomerac
  SIGNAL sF: std_logic_vector(3 DOWNTO 0);
BEGIN

  eCELL_0:CELL PORT MAP (iA=>iA(0),  iB=>iB(0), iC=>iSEL(0),
                         iSEL=>iSEL, oS=>sF(0), oC=>sC(0));
  eCELL_1:CELL PORT MAP (iA=>iA(1),  iB=>iB(1), iC=>sC(0),
                         iSEL=>iSEL, oS=>sF(1), oC=>sC(1));
  eCELL_2:CELL PORT MAP (iA=>iA(2),  iB=>iB(2), iC=>sC(1),
                         iSEL=>iSEL, oS=>sF(2), oC=>sC(2));
  eCELL_3:CELL PORT MAP (iA=>iA(3),  iB=>iB(3), iC=>sC(2),
                         iSEL=>iSEL, oS=>sF(3), oC=>sC(3));

  ePOMERAC:POMERAC PORT MAP (iH=>sF, iSEL=>iSEL, oH=>oH);

  oC <= sC(3);

END ARH_ALJ02;
```

Isto kao i u prethodnom VHDL kodu, ulančavanjem isprojektovanih modula formira se logička šema celokupne ALJ, Slika 8.19.



Slika 8.19: Logička šema tražene ALJ

Simulacijom rada tako formirane ALJ proverava se njena funkcionalnost, Slika 8.20.



Slika 8.20: Simulacija rada realizovane ALJ

Vremenski dijagram je podeljen na 2 dela, na osnovu vrednosti selekcione promenljive `iSEL[2]`. Vektor `sF` predstavlja međurezultat, odnosno izlaz iz aritmetičkog dela ALJ i ulaz u pomerač. Izvršenje odgovarajuće aritmetičko-logičke operacije zavisi od vrednosti selekcionih promenljivih `iSEL[1..0]` dok je vrednost selekcione promenljive `iSEL[2]=0`. Za `iSEL[2]=1` ovaj vektor sadrži vrednost ulaznog operanda A.

U prvom je delu, do 400 ns, `iSEL[2]=0`, pa se na osnovu postavke zadatka na izlazu ALJ (vektor `oH`) daje vrednost rezultata odabrane aritmetičko-logičke operacije. Drugi deo, od 400ns do 800ns, selekciona promenljiva `iSEL[2]` ima vrednost `1` što na izlazu ALJ daje rezultat `sF` pomeren u levo za jedno mesto, u desno za jedno mesto ili nulu u skladu sa postavkom zadatka.

8.4 ZADATAK:

ALJ iz prethodnog zadatka realizovati pomoću VHDL jezika za opis fizičke arhitekture, koristeći opis pomoću vektora.

REŠENJE:

Sinteza VHDL opisa tražene ALJ se može uraditi direktno na osnovu postavke zadatka, odnosno na osnovu traženih operacija koje prikazuje Tabela 8.4. U nastavku je prikazano jedno rešenje koje je formirano sa jednim procesom koji na osnovu selektujućih signala izvršava odgovarajuću operaciju.

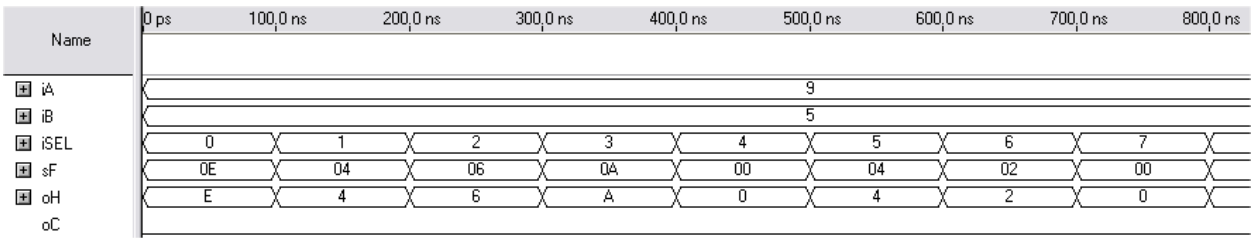
```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;

ENTITY ALJ02 IS PORT(
  iA, iB: IN  std_logic_vector(3 DOWNTO 0);
  iSEL:   IN  std_logic_vector(2 DOWNTO 0);
  oH:     OUT std_logic_vector(3 DOWNTO 0);
  oC:     OUT std_logic);
END ALJ02;
```

```
ARCHITECTURE ARH_ALJ02 OF ALJ02 IS
  -- rezultat operacije, ulaz u pomerac
  -- prosiren za jedan bit gde ce se smestiti
  -- izracunati izlazni prenos
  SIGNAL sF: unsigned(4 DOWNTO 0);
BEGIN
  PROCESS (iA, iB, iSEL)
    -- operandi koji su prosireni za jedan bit
    -- zbog racunanja izlaznog prenosa
    VARIABLE vA, vB: unsigned(4 DOWNTO 0);
  BEGIN
    vA := unsigned('0' & iA); -- operand A prosiren za jedan bit
    vB := unsigned('0' & iB); -- operand B prosiren za jedan bit

    CASE iSEL IS
      WHEN "000"  => sF <= vA + vB;
      WHEN "001"  => sF <= vA - vB;
      WHEN "010"  => sF <= ('0' & NOT(vA(3 DOWNTO 0)));
      WHEN "011"  => sF <= vA + 1;
      WHEN "101"  => sF <= ("00" & vA(3 DOWNTO 1));
      WHEN "110"  => sF <= ('0' & vA(2 DOWNTO 0) & '0');
      WHEN OTHERS => sF <= "00000";
    END CASE;
  END PROCESS;

  -- formiranje izlaza ALJ
  oH <= std_logic_vector(sF(3 DOWNTO 0));
  oC <= std_logic(sF(4));
END ARH_ALJ02;
```

Pošto su ulazni operandi (`iA` i `iB`) i rezultat aritmetičko-logičke operacije (`oH`) predstavljeni tipom `std_logic_vector`, neophodna je konverzija ulaznih operanada u tip nad kojim je moguće izvršiti aritmetičku operaciju u skladu sa VHDL standardom, npr tip `unsigned`. Odgovarajuća konverzija tipova je urađena u samom telu procesa formiranjem promenljivih `vA` i `vB`.

Istom prilikom su te promenljive proširene sa gornje strane sa dodatnim bitom vrednosti '0', radi generisanja signala izlaznog prenosa. Sve operacije realizovane u okviru procesa se izvršavaju nad tim promenljivima. Rezultat je prodstavljen signalom `sF` koji je takođe tipa `unsigned`. Zbog toga je prilikom formiranja izlaza ALJ potrebna konverzija u tip `std_logic_vector`, odnosno `std_logic` za rezultat aritmetičko-logičke (`oH`) operacije i izlazni prenos respektivno (`oC`).

Slika 8.21 prikazuje simulaciju rada realizovane ALJ.



Slika 8.21: Simulacija rada realizovane ALJ

8.5 ZADATAK:

Upotrebom standardnih metoda minimizacije isprojektovati ALJ računarskog sistema za dobijanje operacija koje prikazuje sledeća tabela (Tabela 8.16):

S_2	S_1	S_0	C_0	Izlaz	Funkcija
0	0	0	0	F = A	prenos
0	0	0	1	F = A + 1	uvećanje A
0	0	1	0	F = A + B	sabiranje
0	0	1	1	F = A + B + 1	sabiranje
0	1	0	0	F = A – B – 1	oduzimanje
0	1	0	1	F = A – B	oduzimanje
0	1	1	0	F = A – 1	umanjenje A
0	1	1	1	F = A	prenos
1	0	0	0	$F = A \vee B$	ILI
1	0	1	0	$F = A \oplus B$	XILI
1	1	0	0	$F = A \wedge B$	I
1	1	1	0	$F = \overline{A}$	NE

Tabela 8.9: Operacije ALJ u zavisnosti od selekcionih promenljivih

Isprojektovanu ALJ realizovati formiranjem odgovarajuće šeme logičkih elemenata.

REŠENJE:

Prvi korak kod projektovanja ALJ je projektovanje punog sabirača (videti prethodne zadatke).

Nakon toga sledi projektovanje kombinacione mreže koja formira odgovarajuće ulaze u puni sabirač u zavisnosti od odabrane operacije. U ovom slučaju treba uočiti da tražena ALJ, za razliku od ALJ iz prethodnih zadatka, ima dodatni ulaz označen sa C_0 koji predstavlja ulazni prenos nultog stepena ALJ. Time se omogućava kaskadno povezivanje ALJ sa ciljem dobijanja ALJ za izvršavanje operacija nad većim brojevima (npr. dve četvorobitne ALJ se mogu povezati tako da formiraju osmobitnu ALJ). Prilikom projektovanja takve ALJ ulazni prenos treba posmatrati kao još jedan signal za odabir operacije koja se izvršava.

Ako su A_i i B_i jedan razred (bit) operanda, a X_i, Y_i i C_{ini} ulazi u sabirač, može se definisati tabela na osnovu koje se vrši projektovanje tražene kombinacione mreže (Tabela 8.10).

S_2	S_1	S_0	C_0	C_{ini}	X_i	Y_i	F
0	0	0	0	C_{outi-1}	A_i	0	A
0	0	0	1	C_{outi-1}	A_i	0	A + 1
0	0	1	0	C_{outi-1}	A_i	B_i	A + B
0	0	1	1	C_{outi-1}	A_i	B_i	A + B + 1
0	1	0	0	C_{outi-1}	A_i	$\overline{B_i}$	A – B - 1
0	1	0	1	C_{outi-1}	A_i	$\overline{B_i}$	A - B
0	1	1	0	C_{outi-1}	A_i	1	A - 1
0	1	1	1	C_{outi-1}	A_i	1	A
1	0	0	0	0	$A_i + B_i$	0	$A \vee B$
1	0	1	0	0	A_i	B_i	$A \oplus B$
1	1	0	0	0	$A_i + \overline{B_i}$	$\overline{B_i}$	$A \wedge B$
1	1	1	0	0	A_i	1	$\overline{A_i}$

Tabela 8.10: Tabela ulaznih funkcija u pun sabirač

Prilikom određivanja ulaza u sabirač za izvršenje logičkih operacija uzeta je u obzir činjenica da se ulaz Y_i menja na svake dve vrednosti ulaznih selekcionih promenljivih. Zbog toga je za izvršenje logičke operacije $A \vee B$ postavljeno da ulaz Y_i bude 0, dok je za logičku operaciju $A \wedge B$ ulaz Y_i postavljen na $\overline{B_i}$. Za obe logičke operacije je ulaz X_i punog sabirača određen analitičkom metodom korišćenom u prvom zadatku ovog poglavlja.

Iz tabele se može uočiti da se u izrazu za X_i za sve vrednosti selekcionih promenljivih pojavljuje A_i dok se B_i pojavljuje samo uz selekcione promenljive $S_2 \cdot \overline{S_1} \cdot \overline{S_0}$ a $\overline{B_i}$ uz $\overline{S_2} \cdot \overline{S_1} \cdot S_0$. C_{ini} ima vrednost C_{outi-1} uz selekcionu promenljivu $\overline{S_2}$ tj. vrednost 0 uz S_2, što omogućava direktno pisanje izraza:

$$X_i = A_i + S_2 \cdot \overline{S_1} \cdot \overline{S_0} \cdot B_i + S_2 \cdot S_1 \cdot \overline{S_0} \cdot \overline{B_i}$$

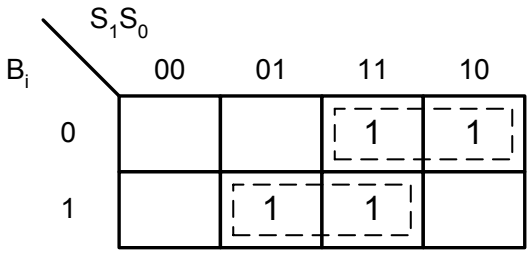
$$C_{ini} = \overline{S_2} \cdot C_{outi-1}$$

Potrebno je formirati pomoćnu tabelu radi jednostavnijeg utvrđivanja funkcije Y_i.

S_1	S_0	Y_i
0	0	0
0	1	B_i
1	0	$\overline{B_i}$
1	1	1

Tabela 8.11: Pomoćna tabela za utvrđivanje funkcije Y_i

Na osnovu tabele se može formirati Karnoova karta za minimizaciju funkcije Y_i, Slika 8.22.

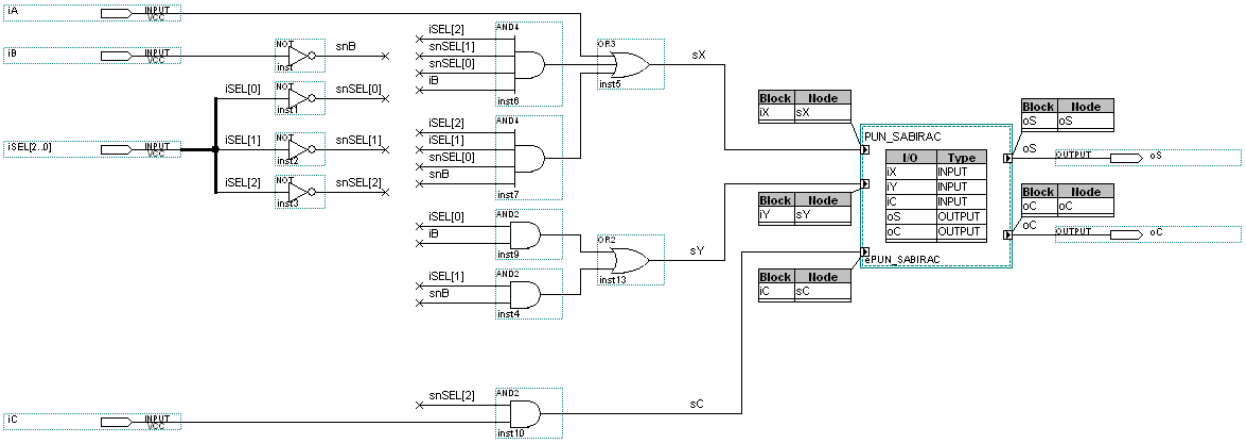


Slika 8.22: Karnoova karta za minimizaciju funkcije Y_i

Minimizacijom se dobija sledeći izraz za Y_i.

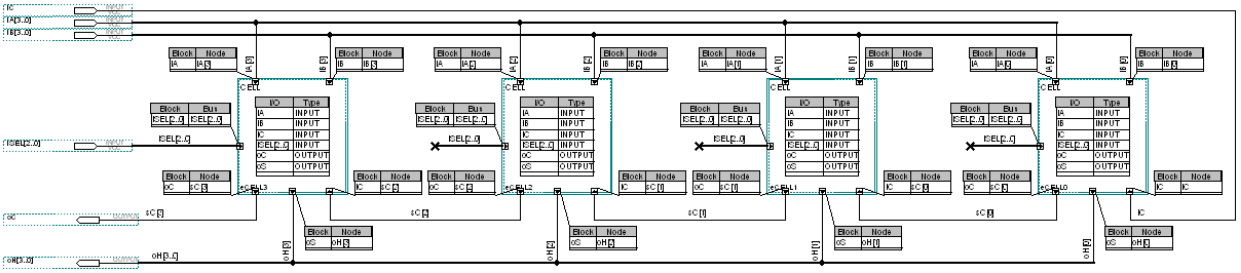
$$Y_i = S_0 \cdot B_i + S_1 \cdot \overline{B_i}$$

Ovim je završeno projektovanje jednog razreda ALJ i može se pristupiti projektovanju logičke šeme, Slika 8.23.



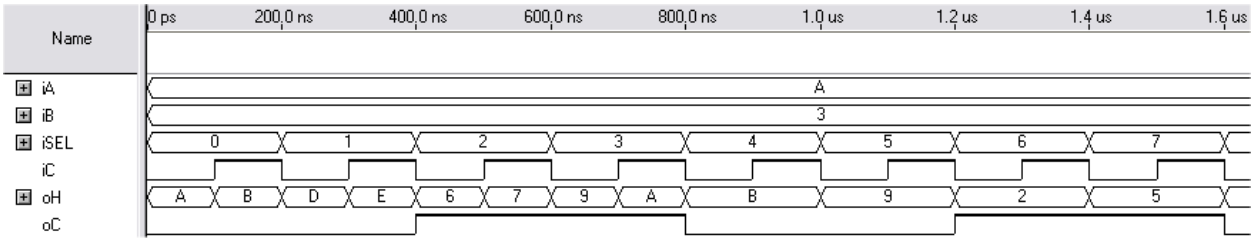
Slika 8.23: Logička šema jednobitne ćelije ALJ

Pošto tražena ALJ nema pomerač, sledeći korak je formiranje logičke šeme kompletne ALJ ulančavanjem realizovanih jednobitnih ćelija, Slika 8.24.



Slika 8.24: Logička šema ALJ

Provera funkcionalne ispravnosti isprojektovane ALJ se izvršava odgovarajućom simulacijom rada. Jedan vremenski dijagram rada tražene ALJ prikazuje Slika 8.25. Na dijagramu je prikazano izvršenje svih operacija ALJ, Tabela 8.16, za jedan par ulaznih operanada.



Slika 8.25: Vremenski dijagram rada projektovane ALJ

Na osnovu istih pricipa može se formirati ekvivalntan VHDL kod tražene aritmetičko logičke jedinice

8.6 ZADATAK:

ALJ iz prethodnog zadatka realizovati pomoću VHDL jezika za opis fizičke arhitekture koristeći opis pomoću vektora.

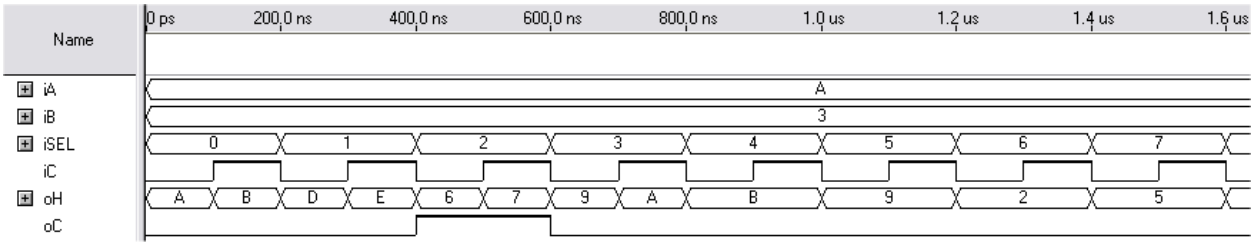
REŠENJE:

Sinteza VHDL opisa tražene ALJ se može uraditi direktno na osnovu postavke zadatka, odnosno na osnovu traženih operacija koje prikazuje Tabela 8.16. U nastavku je prikazano jedno rešenje koje je formirano sa jednim procesom koji na osnovu selektujućih signala izvršava odgovarajuću operaciju. Formiranje izlaznog prenosa je rešeno sa proširivanjem ulaznih operanada nulom sa strane bita najveće važnosti. Oduzimanje je realizovano preko sabiranja sa drugim komplementom.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;

ENTITY ALJ03 IS PORT(
          -- ulazni operandi
  iA, iB: IN  unsigned(3 DOWNTO 0);
          -- ulazni prenos
  iC:     IN  std_logic;
          -- vektor za odabir operacije
  iSEL:   IN  std_logic_vector(2 DOWNTO 0);
          -- rezultat
  oH:     OUT unsigned(3 DOWNTO 0);
          -- izlazni prenos
  oC:     OUT std_logic );
END ALJ03;
```

```
ARCHITECTURE ARH_ALJ03 OF ALJ03 IS
  -- vektor za selekciju operacije koji objedinjuje
  -- ulazni vektor i signal ulaznog prenosa
  SIGNAL sSEL: std_logic_vector(3 DOWNTO 0);

  -- operandi koji su prosireni za jedan bit
  -- zbog racunanja izlaznog prenosa
  SIGNAL sA, sB: unsigned(4 DOWNTO 0);

  -- rezultat operacije, ulaz u pomerac
  -- prosiren za jedan bit gde ce se smestiti
  -- izracunati izlazni prenos
  SIGNAL sF:  unsigned(4 DOWNTO 0);
BEGIN

  sSEL <= (iSEL & iC); -- vektor za selekciju operacije
  sA   <= ('0' & iA);  -- operand A prosiren za jedan bit
  sB   <= ('0' & iB);  -- operand B prosiren za jedan bit

  PROCESS(sSEL, iA, iB, sA, sB) BEGIN
    CASE sSEL IS
      WHEN "0000" => sF <= sA;
      WHEN "0001" => sF <= sA + 1;
      WHEN "0010" => sF <= sA + sB;
      WHEN "0011" => sF <= sA + sB + 1;
      WHEN "0100" => sF <= sA + ('0' & NOT(iB));
      WHEN "0101" => sF <= sA + ('0' & NOT(iB)) + 1;
      WHEN "0110" => sF <= sA - 1;
      WHEN "0111" => sF <= sA;
      WHEN "1000" => sF <= sA OR  sB;
      WHEN "1001" => sF <= sA OR  sB;
      WHEN "1010" => sF <= sA XOR sB;
      WHEN "1011" => sF <= sA XOR sB;
      WHEN "1100" => sF <= sA AND sB;
      WHEN "1101" => sF <= sA AND sB;
      WHEN OTHERS => sF <= ('0' & NOT(iA));
    END CASE;
  END PROCESS;

  -- formiranje izlaza ALJ
  oH <= sF(3 DOWNTO 0);
  oC <= std_logic(sF(4));
END ARH_ALJ03;
```

Simulaciju rada isprojektovane ALJ prikazuje vremenski dijagram na sledećoj slici (Slika 8.26).



Slika 8.26: Vremenski dijagram rada isprojektovane ALJ

8.7 ZADATAK:

Isprojektovati kombinacionu mrežu koja izvršava operaciju množenja dva neoznačena četvorobitna broja metodom koja simulira "ručno množenje" binarnih brojeva. Prikazati rešenje primenom VHDL jezika za opis fizičke arhitekture.

REŠENJE:

Procedura množenja se realizuje formiranjem parcijalnih proizvoda za svaki bit množitelja. Parcijalni proizvodi se definišu na osnovu svakog bita množitelja. Ako dati bit ima vrednost jedan, tada je parcijalni proizvod množenik, u suprotnom parcijalni proizvod ima vrednost nula. Nakon formiranja svih parcijalnih proizvoda izvršava se njihovo sabiranje čime se formira krajnji proizvod. Za ovu operaciju se svaki uzastopni parcijalni proizvod pomera za jednu poziciju ulevo u odnosu na prethodni parcijalni proizvod. Na ovaj način se množenjem dva broja od n bita dobija proizvod od 2n bita.

Opisana procedura se ilustruje na množenju dva četvorobitna broja:

				1	0	1	1	Množenik (11)
			×	1	1	0	1	Množitelj (13)
				1	0	1	1	
			0	0	0	0		Parcijalni
		1	0	1	1			proizvodi
	1	0	1	1				
1	0	0	0	1	1	1	1	Proizvod (143)
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	

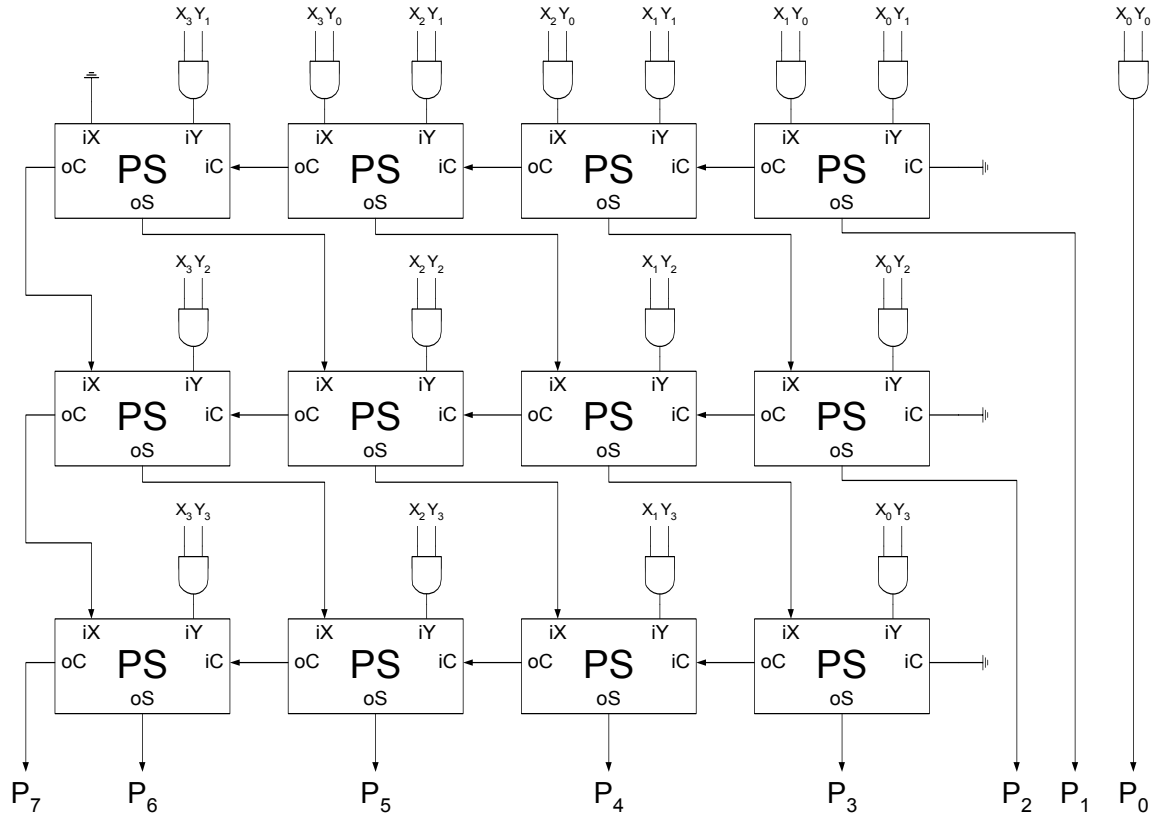
Uopštena šema množenja dva četvorobitna broja je sledeća:

			x_3	x_2	x_1	x_0	Množenik (X)
		×	y_3	y_2	y_1	y_0	Množitelj (Y)
			x_3y_0	x_2y_0	x_1y_0	x_0y_0	
		x_3y_1	x_2y_1	x_1y_1	x_0y_1		Parcijalni
	x_3y_2	x_2y_2	x_1y_2	x_0y_2			proizvodi
+ x_3y_3	x_2y_3	x_1y_3	x_0y_3				
$K_6 \cdot 2^6 + K_5 \cdot 2^5 + K_4 \cdot 2^4 + K_3 \cdot 2^3 + K_2 \cdot 2^2 + K_1 \cdot 2^1 + K_0 \cdot 2^0$							Proizvod (P)

Za množenje četvorobitnih brojeva potrebno je formirati mrežu punih sabirača. Mreža sadrži tri stepena od po četiri puna sabirača. Svaki stepen sabirača je povezan tako da formira četvorobitni paralelni sabirač (izlazni prenos prvog sabirača je vezan na ulazni prenos drugog sabirača itd.). Na ulaze punih sabirača iz prvog stepena dovode se proizvodi odgovarajućih bita množenika sa bitovima 0 i 1 množitelja. Rezultat sabirača iz prvog stepena i proizvodi bita množenika sa bitom 2 množitelja se dovode na ulaze sabirača iz drugog stepena. Slično, sabirači iz

trećeg stepena imaju za ulaze rezultat sabiranja iz drugog stepena i proizvod množenika sa bitom 3 množitelja. Svaki stepen je pomeren za jedno mesto ulevo u odnosu na prethodni, tako da izlaz prvog sabirača iz svakog stepena direktno formira odgovarajući bit proizvoda. Na ulaz poslednjeg sabirača iz drugog i tećeg stepena dovodi se izlazni prenos poslednjeg sabirača iz prethodnog stepena.

Blok šemu ovakve kombinacione mreže prikazuje Slika 8.27.



Slika 8.27: Blok šema kombinacione mreže za množenje dva četvorobitna broja

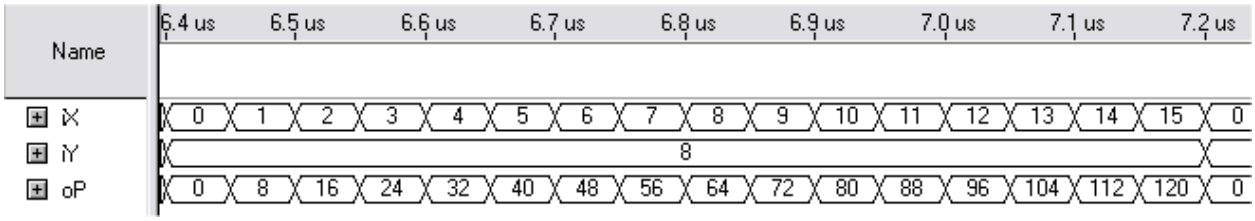
Na osnovu formirane blok šeme kombinacione mreže za množenje dva četvorobitna broja može se isprojektovati odgovarajući VHDL kod:

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY MNOZAC IS PORT(
  iX, iY: IN  std_logic_vector(3 DOWNTO 0);
  oP:     OUT std_logic_vector(7 DOWNTO 0) );
END MNOZAC;
ARCHITECTURE ARH_MNOZAC OF MNOZAC IS

  -- opis sprege komponente punog sabiraca
  -- na osnovu koje se realizuje mnozac
  -- kao kombinaciona mreza
  COMPONENT PS PORT (
    iX, iY, iC: IN  std_logic;
    oS, oC:     OUT std_logic );
  END COMPONENT;
```

```
    -- konstanta nula
    CONSTANT cZERO: std_logic := '0';

    -- vektori koji sadrze rezultate mnozenja
    -- pojedinacnih bita vektora X sa odgovarajucim bitom vektora Y
    SIGNAL sBIT0, sBIT1, sBIT2, sBIT3: std_logic_vector(3 DOWNTO 0);
    -- rezultati i izlazni prenosi prvog stepena sabiraca
    SIGNAL sP00, sP01, sP02, sP03, sC00, sC01, sC02, sC03: std_logic;
    -- rezultati i izlazni prenosi drugog stepena sabiraca
    SIGNAL sP10, sP11, sP12, sP13, sC10, sC11, sC12, sC13: std_logic;
    -- rezultati i izlazni prenosi treceg stepena sabiraca
    SIGNAL sP20, sP21, sP22, sP23, sC20, sC21, sC22, sC23: std_logic;
BEGIN

    -- mnozenje sa bitom nula vektora Y
    sBIT0(0) <= iX(0) AND iY(0);
    sBIT0(1) <= iX(1) AND iY(0);
    sBIT0(2) <= iX(2) AND iY(0);
    sBIT0(3) <= iX(3) AND iY(0);

    -- mnozenje sa bitom jedan vektora Y
    sBIT1(0) <= iX(0) AND iY(1);
    sBIT1(1) <= iX(1) AND iY(1);
    sBIT1(2) <= iX(2) AND iY(1);
    sBIT1(3) <= iX(3) AND iY(1);

    -- mnozenje sa bitom dva vektora Y
    sBIT2(0) <= iX(0) AND iY(2);
    sBIT2(1) <= iX(1) AND iY(2);
    sBIT2(2) <= iX(2) AND iY(2);
    sBIT2(3) <= iX(3) AND iY(2);

    -- mnozenje sa bitom tri vektora Y
    sBIT3(0) <= iX(0) AND iY(3);
    sBIT3(1) <= iX(1) AND iY(3);
    sBIT3(2) <= iX(2) AND iY(3);
    sBIT3(3) <= iX(3) AND iY(3);

    -- prvi stepen sabiraca
    ePS00: PS PORT MAP(iX=>sBIT0(1), iY=>sBIT1(0), iC=>cZERO,
                       oS=>sP00,     oC=>sC00);
    ePS01: PS PORT MAP(iX=>sBIT0(2), iY=>sBIT1(1), iC=>sC00,
                       oS=>sP01,     oC=>sC01);
    ePS02: PS PORT MAP(iX=>sBIT0(3), iY=>sBIT1(2), iC=>sC01,
                       oS=>sP02,     oC=>sC02);
    ePS03: PS PORT MAP(iX=>cZERO,    iY=>sBIT1(3), iC=>sC02,
                       oS=>sP03,     oC=>sC03);

    -- drugi stepen sabiraca
    ePS10: PS PORT MAP(iX=>sP01,     iY=>sBIT2(0), iC=>cZERO,
                       oS=>sP10,     oC=>sC10);
    ePS11: PS PORT MAP(iX=>sP02,     iY=>sBIT2(1), iC=>sC10,
                       oS=>sP11,     oC=>sC11);
    ePS12: PS PORT MAP(iX=>sP03,     iY=>sBIT2(2), iC=>sC11,
                       oS=>sP12,     oC=>sC12);
    ePS13: PS PORT MAP(iX=>sC03,     iY=>sBIT2(3), iC=>sC12,
                       oS=>sP13,     oC=>sC13);
```

```
    -- treci stepen sabiraca
    ePS20: PS PORT MAP(iX=>sP11,     iY=>sBIT3(0), iC=>cZERO,
                       oS=>sP20,     oC=>sC20);
    ePS21: PS PORT MAP(iX=>sP12,     iY=>sBIT3(1), iC=>sC20,
                       oS=>sP21,     oC=>sC21);
    ePS22: PS PORT MAP(iX=>sP13,     iY=>sBIT3(2), iC=>sC21,
                       oS=>sP22,     oC=>sC22);
    ePS23: PS PORT MAP(iX=>sC13,     iY=>sBIT3(3), iC=>sC22,
                       oS=>sP23,     oC=>sC23);

    -- formiranje izlaznog vektora
    oP(0) <= sBIT0(0);
    oP(1) <= sP00;
    oP(2) <= sP10;
    oP(3) <= sP20;
    oP(4) <= sP21;
    oP(5) <= sP22;
    oP(6) <= sP23;
    oP(7) <= sC23;

END ARH_MNOZAC;
```

Slika 8.28 prikazuje simulaciju rada množača, gde se broj 8 množi sa svih 16 mogućih vrednosti.



Slika 8.28: Simulacija rada realizovanog množača

8.8 ZADATAK:

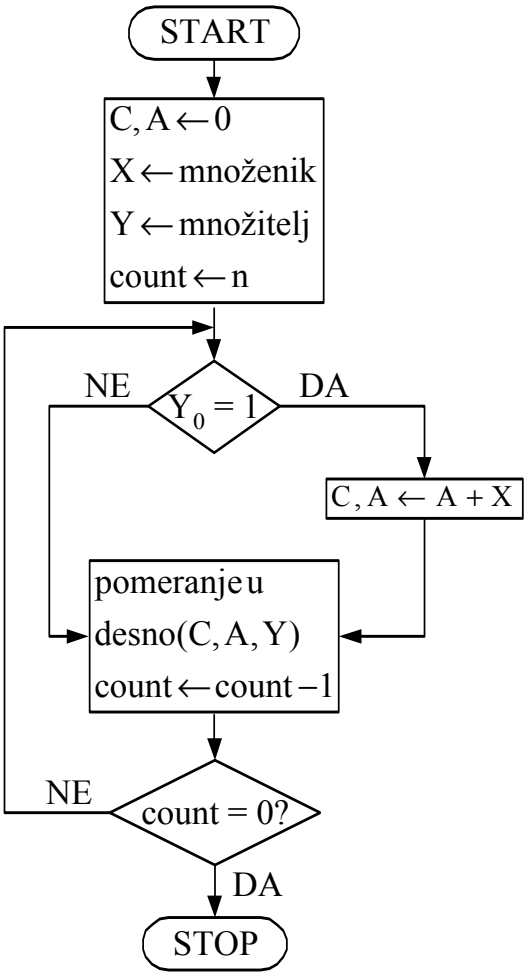
Izvršiti sintezu digitalnog sistema za množenje celih neoznačenih brojeva prema algoritmu koji prikazuje Slika 8.29. Digitalni sistem treba projektovati za potrebe množenja četvorobitnih brojeva.

U rešenju treba prikazati blok šemu digitalnog sistema, graf prelazaka stanja generatora upravljačkih signala kao i VHDL kod kompletnog digitalnog sistema. Prikazati funkcionisanje isprojektovanog digitalnog sistema na primeru množenja brojeva 11 i 13.

REŠENJE:

Prikazani blok dijagram algoritma izvršava "ručno"množenje binarnih brojeva. Pri tome su množenik i množitelj smešteni u registrima X i Y respektivno. Rezultat množenja se akumulira u registru A. Prilikom formiranja rezultata, uvažava se vrednost trenutnog bita množitelja tako što se operacija sabiranja izvršava samo ako dati bit ima vrednost jedan. U suprotnom slučaju bi se

izvršavalo sabiranje sa nulom, pa se zbog toga izostavlja izvršenje ove operacije. Pomeranje udesno za jednu poziciju se izvršava za obe vrednosti trenutnog bita množitelja.

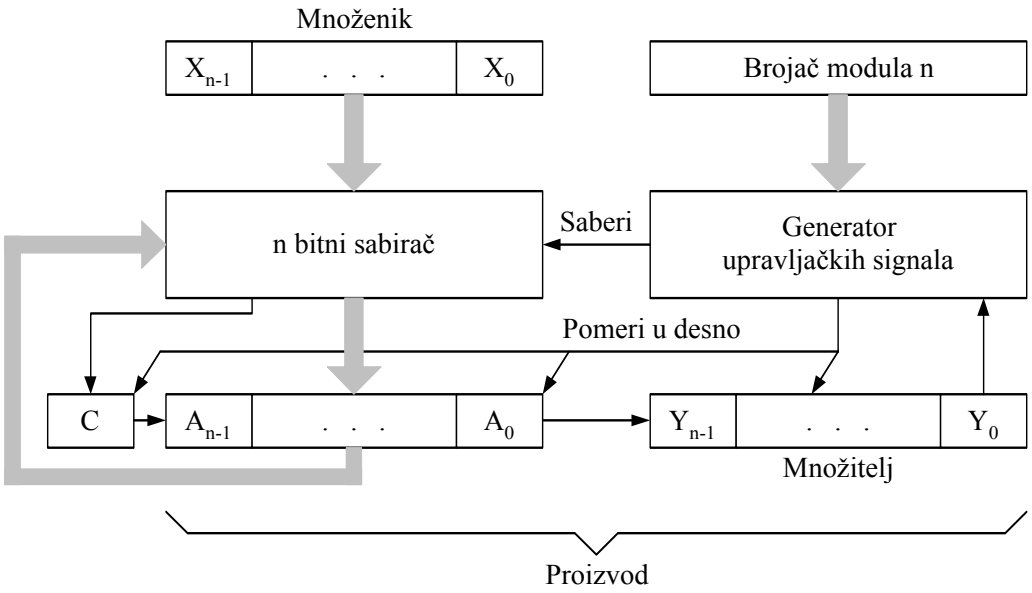


Slika 8.29: Blok dijagram algoritma za množenje celih neoznačenih brojeva

Prema tome, za realizaciju traženog sistema potrebna su 3 četvorobitna registra (X, Y i A) kao i jednobitni registar C za smeštanje izlaznog prenosa sabirača. Takođe, potreban je i brojač modula 4 koji određuje koji se trenutno bit množitelja analizira. Potreban je i jedan modul koji će generisati odgovarajuće upravljačke signale za upis u registre, dozvolu pomeranja i dozvolu brojanja. Blok šemu ovakvog digitalnog sistema prikazuje Slika 8.30.

Nakon inicijalizacije registara i brojača, dobijeni digitalni sistem funkcioniše na sledeći način. Generator upravljačkih signala čita stanje bita 0 množitelja Y. Ako dati bit ima vrednost jedan tada se izvršava sabiranje množenika X sa stanjem akumulatorskog registra A, pri čemu se rezultat upisuje u akumulatorski registar A i registar izlaznog prenosa C. Nakon toga se izvršava pomeranje u desno za jednu poziciju svih bita registara C, A i Y, tako da sadržaj registra C ide na poziciju A_{n-1}, A_0 ide u Y_{n-1} i bit Y_0 se gubi. Takođe, smanjuje se vrednost brojača za jedan čime se započinje novi ciklus za drugi bit množitelja koji je sada smešten na poziciji Y_0. U slučaju da analizirani bit množitelja ima vrednost nula ne izvršava se sabiraje

već samo pomeranje. Ovaj proces se ponavlja za sve bite množitelja, odnosno dok brojač ne poprimi vrednost nula. Dobijeni proizvod od 2·n=8 bita se nakon završetka izvršenja algoritma nalazi u registrima A i Y.



Slika 8.30: Blok šema digitalnog sistema za izvršenje traženog algoritma

U zadatku su zadate vrednosti sa kojima treba ilustrovati proceduru množenja. Množenik ima vrednost $11_{DEC}=1011_{BIN}$, dok je množitelj $13_{DEC}=1101_{BIN}$. Njihov proizvod je $143_{DEC}=10001111_{BIN}$. Prema tome, na kraju izvršenja algoritma u registru A treba da je vrednost 1000_{BIN}, a u registru Y vrednost 1111_{BIN}. Za date vrednosti množenika i množitelja algoritam funcioniše na sledeći način:

C	A	Y	X		
0	0000	1101	1011	Inicijalne vrednosti	
0	1011	1101	1011	Sabiranje	Prvi
0	0101	1110	1011	Pomeranje	ciklus
0	0010	1111	1011	Pomeranje	Drugi ciklus
0	1101	1111	1011	Sabiranje	Treći
0	0110	1111	1011	Pomeranje	ciklus
1	0001	1111	1011	Sabiranje	Četvti
0	1000	1111	1011	Pomeranje	ciklus

Nakon prethodne analize problema, pristupa se formiranju VHDL opisa traženog digitalnog sistema. U ovom slučaju će se traženi sistem realizovati modularno, projektovanjem zasebnih entiteta sa opisom arhitektura registra, pomeračkog registra, brojača, sabirača i generatora upravljačkih signala. Na kraju će se svi navedeni moduli povezati u celinu koju prikazuje Slika 8.30. Radi lakše

realizacije povezivanja modula, svi signali za povezivanje modula su tipa STD_LOGIC.

REGISTAR:

Prvi realizovani modul je registar za smeštanje množenika. Spregu ovog modula čine sledeći signali:

- iCLK takt signal,
- inCLR signal za brisanje sadržaja registra, aktivan na niskom nivou i sinhron sa signalom takta,
- iCE dozvola upisa u registar (ako je ovaj signal na visokom nivou u registar se na rastuću ivicu takt signala upisuje vrednost prisutna na ulaznom vektoru podataka iD),
- iD ulazni vektor podataka i
- oQ stanje registra,

Sa ciljem formiranja generalizovanog registra sa prizvoljnim brojem bita uvodi se generički parametar pWIDTH koji označava broj bita registra koji se instancira. Uvođenjem ovog parametra registar će imati bite sa indeksima 0 (bit najmanje važnosti) do pWIDTH-1 (bit najveće važnosti), što ukupno čini pWIDTH bita. Pretpostavlja se da instancirani registar ima četiri bita.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY REG IS
  GENERIC (
    pWIDTH: integer := 4    -- pretpostavljeni broj bita je 4
  );
  PORT (
    iCLK, inCLR: IN  STD_LOGIC;
    iCE: IN  STD_LOGIC;
    iD:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oQ:  OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
  );
END REG;

ARCHITECTURE ARH_REG OF REG IS
  -- stanje registra
  SIGNAL sREG: STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
BEGIN
 PROCESS (iCLK) BEGIN
  IF (iCLK'EVENT AND iCLK = '1') THEN
    IF (inCLR = '0') THEN -- sinhrono postavljanje pocetne vrednosti
     sREG <= (OTHERS => '0');
    ELSE
     IF (iCE = '1') THEN -- upis u registar
      sREG <= iD;
     ELSE
      sREG <= sREG;
```

```
      END IF;
     END IF;
    END IF;
  END PROCESS;

  -- preslikavanje stanja registra na izlazni vektor
  oQ <= sREG;

END ARH_REG;
```

POMERAČKI REGISTAR:

Modul sa opisom pomeračkog registra takođe ima generički parametar `pWIDTH` koji definiše broj bita registra koji se instancira. Njegova pretpostavljena vrednost je 4, čime se pretpostavlja da instancirani pomerački registar ima četiri bita. Smer pomeranja je u desno.

Spregu modula pomeračkog registra čine sledeći signali:

- `iCLK` takt signal,
- `inCLR` signal za brisanje sadržaja registra, aktivan na niskom nivou i sinhron sa signalom takta,
- `iDSR` *Data Shift Right*, vrednost koja se upisuje u bit najveće važnosti registra prilikom pomeranja u desno,
- `iLOAD` dozvola upisa u registar (ako je ovaj signal na visokom nivou u registar se na rastuću ivicu takt signala upisuje vrednost prisutna na ulaznom vektoru podataka `iD`),
- `iSE` dozvola pomeranja sadržaja registra u desno aktivna na visokom nivou,
- `iD` ulazni vektor podataka i
- `oQ` stanje registra,

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY SHIFT_REG IS
  GENERIC (
    pWIDTH: integer := 4    -- pretpostavljeni broj bita je 4
  );
  PORT (
    iCLK, inCLR: IN  STD_LOGIC;
    iDSR, iLOAD, iSE: IN  STD_LOGIC;
    iD: IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oQ: OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
  );
END SHIFT_REG;
ARCHITECTURE ARH_SHIFT_REG OF SHIFT_REG IS
  -- stanje registra
  SIGNAL sREG: STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
BEGIN

 PROCESS (iCLK) BEGIN
  IF (iCLK'EVENT AND iCLK = '1') THEN
```

```
    IF (inCLR = '0') THEN   -- sinhrono postavljanje pocetne vrednosti
     sREG <= (OTHERS => '0');
    ELSE
     IF (iLOAD = '1') THEN -- upis u registar
      sREG <= iD;
     ELSE
      IF (iSE = '1') THEN -- pomeranje u desno
       sREG(pWIDTH-1) <= iDSR;
       FOR i IN 0 TO (pWIDTH-2) LOOP
        sREG(i) <= sREG(i+1);
       END LOOP;
      ELSE                -- zadrzavanje stanja
       sREG <= sREG;
      END IF;
     END IF;
    END IF;
   END IF;
  END PROCESS;

  -- preslikavanje stanja registra na izlazni vektor
  oQ <= sREG;

END ARH_SHIFT_REG;
```

Pošto je sintetizovan pomerački registar sa generičkim parametrom koji određuje broj bita, operacija pomeranja je realizovana upotrebom `FOR` petlje:

```
sREG(pWIDTH-1) <= iDSR;
FOR i IN 0 TO (pWIDTH-2) LOOP
  sREG(i) <= sREG(i+1);
END LOOP;
```

Pri tome se kreće od bita najniže važnosti (bit sa indeksom `0`) koji prima vrednost susednog bita veće važnosti, i tako redom sve do pretposlednjeg bita (bit sa indeksom `pWIDTH-2`) koji prima vrednost bita najveće važnosti (bit sa indeksom `pWIDTH-1`). Stanje na ulaznom signalu `iDSR` se upisuje u bit najveće važnosti.

BROJAČ:

Potrebno je realizovati brojač modula 4 (ili generalizovani brojač modula n) koji generiše izlazni signal koji ukazuje na kraj ciklusa brojanja. Radi lakšeg definisanja kombinacione mreže za detekciju kraja ciklusa brojanja realizovan je brojač na dole, gde je kraj ciklusa kad brojač stigne do vrednosti nula.

Modul brojača ima sledeće sprežne signale:

- `iCLK` takt signal,
- `inRESET` signal za brisanje sadržaja brojača, aktivan na niskom nivou i sinhron sa signalom takta,
- `iEN` dozvola brojanja aktivna na visokom nivou,

- iP dozvola upisa inicijalne vrednosti u brojač (ako je ovaj signal na visokom nivou u brojač se na rastuću ivicu takt signala upisuje vrednost prisutna na ulaznom vektoru podataka iDATA),
- iDATA ulazni vektor podataka,
- oQ stanje brojača i
- oTSE indikacija kraja ciklusa brojanja, aktivna na visokom nivou

Pomoću generičkog parametra pWIDTH je moguće definisati broj bita brojača koji se instancira. Vrednost ovog parametra mora biti u skladu sa širinom vektora podataka koji se upisuje u brojač.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;

ENTITY BROJAC IS
  GENERIC (
    pWIDTH: integer := 4    -- pretpostavljeni broj bita je 4
  );
  PORT (
    iCLK    : IN  STD_LOGIC;
    inRESET,
    iEN, iP : IN  STD_LOGIC;
    iDATA   : IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oQ      : OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oTSE    : OUT STD_LOGIC
  );
END BROJAC;

ARCHITECTURE ARH_BROJAC OF BROJAC IS
  -- stanje brojaca
  SIGNAL sCNT: UNSIGNED(pWIDTH-1 DOWNTO 0);
BEGIN
  -- brojac sa sinhronim resetom
  PROCESS(iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      IF (inRESET = '0') THEN -- sinhroni reset
        sCNT <= (OTHERS => '0');
      ELSE
        IF (iP = '1') THEN  -- dozvoljen paralelni upis?
          sCNT <= UNSIGNED(iDATA);  -- paralelni upis
        ELSE
          IF (iEN = '1') THEN -- dozvoljeno brojanje?
            sCNT <= sCNT - 1; -- brojanje
          END IF;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- formiranje izlaznog signala koji
  -- oznacava kraj ciklusa brojanja
```

```
  PROCESS (sCNT) BEGIN
    IF  (sCNT = 0) THEN
      oTSE <= '1';  -- kraj ciklusa brojanja
    ELSE
      oTSE <= '0';
    END IF;
  END PROCESS;

  -- formiranje izlaznog vektora
  oQ <= STD_LOGIC_VECTOR(sCNT);
END ARH_BROJAC;
```

SABIRAČ:

Sabiranjem dva vektora od n bita dobija se izlazni vektor od n+1 bita. Zbog toga realizovani modul sabirača ima dva ulazna vektora, `iX` i `iY`, od `pWIDTH` bita i jedan izlazni vektor `oZ` koji se sastoji od `pWIDTH+1` bita. `pWIDTH` je generički parametar koji određuje broj bita ulaznih vektora.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;

ENTITY ADDER IS
  GENERIC (
    pWIDTH: integer := 4   -- pretpostavljeni broj bita je 4
  );
  PORT (
    iX, iY:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oZ:      OUT STD_LOGIC_VECTOR(pWIDTH   DOWNTO 0)
  );
END ADDER;

ARCHITECTURE ARH_ADDER OF ADDER IS
  SIGNAL sX, sY: UNSIGNED(pWIDTH-1 DOWNTO 0);
  SIGNAL sZ:     UNSIGNED(pWIDTH   DOWNTO 0);
BEGIN
  -- konvertovanje ulaznih vektora u tip UNSIGNED
  sX <= UNSIGNED(iX);
  sY <= UNSIGNED(iY);

  -- sabiranje, parametri su prosireni nulom zbog
  -- izjednacavanja broja bita sa leve i desne strane
  -- znaka dodele vrednosti signalu
  sZ <= ('0' & sX) + ('0' & sY);

  -- konvertovanje rezultata u tip STD_LOGIC_VECTOR
  oZ <= STD_LOGIC_VECTOR(sZ);
END ARH_ADDER;
```

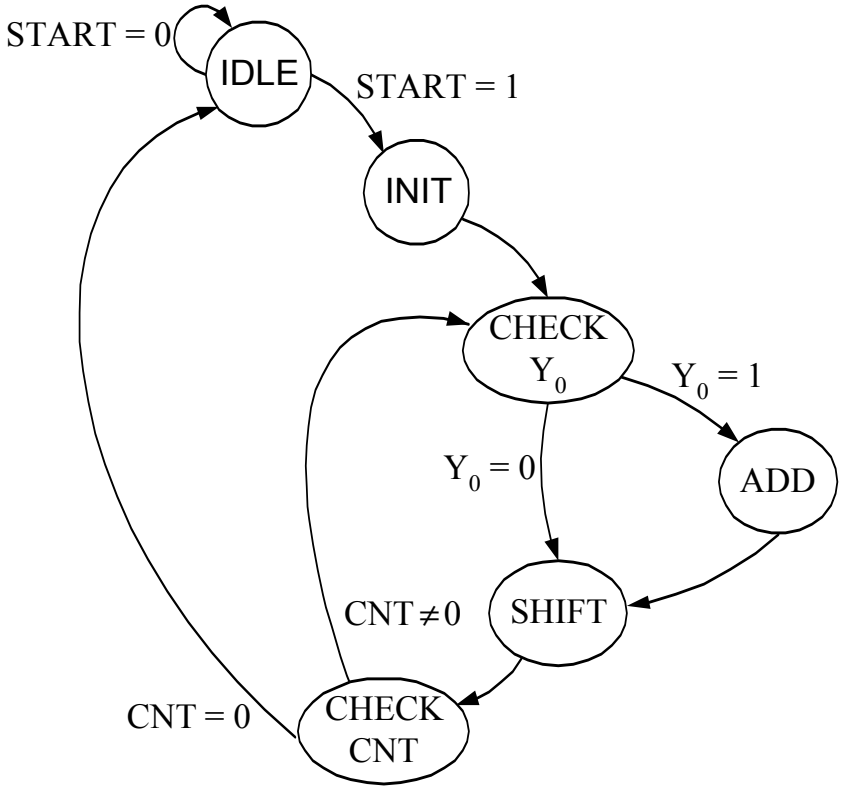
Pošto su svi sprežni signali tipa `STD_LOGIC_VECTOR` potrebno ih je konvertovati u tip `UNSIGNED` pre same realizacije sabirača. Na ulaze sabirača se dovode ulazni vektori prošireni sa nulom na poziciji bita najveće važnosti čime se na ulaz sabirača dovode vektori od `pWIDTH+1` bita. Ovakva realizacija je potrebna

zbog VHDL standarda koji govori da sa obe strane znaka dodele vrednosti signalu vektori moraju imati isti broj bita. Rezultat sabiranja se konvertuje u tip `STD_LOGIC_VECTOR` i prosleđuje na izlazni vektor `oZ`.

Može se primetiti da ovako realizovan sabirač nema bit izlaznog prenosa. Međutim, ovaj bit postoji na poziciji bita najveće važnosti rezultata pošto se na tom mestu sabiraju nule sa kojima su prošireni ulazni parametri i prenos prilikom sabiranja bita najveće važnosti ulaznih parametara. To rezultuje postavljanjem izlaznog prenosa na mesto bita najveće važnosti rezultata sabiranja.

MODUL ZA GENERISANJE UPRAVLJAČKIH SIGNALA:

Ovaj modul treba da na osnovu zadatog blok dijagram algoritma množenja neoznačenih brojeva, Slika 8.29, generiše odgovarajuće upravljačke signale za registre i brojač. Radi lakše implementacije formira se graf prelazaka stanja, Slika 8.31. U svakom stanju se generišu odgovarajući upravljački signali u skladu sa potrebnom operacijom u tom vremenskom trenutku. Ovako definisani upravljački algoritam zapravo predstavlja Murov automat, gde vrednost izlaznih signala zavisi samo od trenutnog stanja, dok prelazi stanja zavise od trenutnog stanja i vrednosti ulaznih signala.



Slika 8.31: Graf prelazaka stanja generatora upravljačkih signala

Automat se inicijalno nalazi u stanju `IDLE`. U slučaju da se na ulazu pojavi impuls za početak izvršenja algoritma, `START=1`, automat prelazi u stanje `INIT` gde se izvršava inicijalizacija stanja svih registara i brojača. Nakon toga prelazi se

u stanje CHECK_Y0 gde se proverava vrednost prvog bita množitelja, Y_0, i odlučuje da li je potrebno izvršiti sabiranje ili ne. Ako je $Y_0=0$ sabiranje nije potrebno pa se odmah prelazi se u stanje SHIFT. U suprotnom slučaju, $Y_0=1$, kada je potrebno sabiranje prelazi se u stanja ADD i SHIFT redom. U stanju SHIFT se izvršava pomeranje registara C, A i Y, kao i dekrementiranje vrednosti brojača. U sledećem stanju, CHECK_CNT, proverava se da li je brojač stigao do kraja ciklusa brojanja, kada se automat vraća u inicijalno stanje IDLE. Ako brojač nije stigao do kraja ciklusa brojanja prelazi se u stanje CHECK_Y0 i postupak se ponavlja do kraja ciklusa brojanja. Tada se završava izvršenje algoritma množenja neoznačenih celih brojeva.

Upravljački automat u ovom slučaju ima sledeće ulazne signale:

- iCLK takt signal,
- inRESET signal za postavljanje automata u inicijalno stanje, aktivan na niskom nivou i sinhron sa signalom takta,
- iSTART impuls za početak izvršenja algoritma množenja,
- iY0 vrednost bita nula množitelja na osnovu kojeg se odlučuje da li će se izvršiti operacija sabiranja ili ne,
- iCNT_0 indikacija kraja ciklusa brojanja brojača ciklusa množenja, aktivna na visokom nivou.

Potrebno je generisati sledeće upravljačke signale:

- oLOAD_CNT impuls za upis početne vrednosti brojača,
- oLOAD_X impuls za upis množioca,
- oLOAD_Y impuls za upis množitelja,
- oLOAD_A impuls za upis rezultata sabiranja u akumulator,
- onRST_A impuls za brisanje sadržaja akumulatora, aktivan na niskom nivou,
- oCE dozvola pomeranja i dozvola brojanja,
- oREADY izlazni signal koji govori da je u toku izvršenje algoritma, ako je njegova vrednost na niskom logičkom nivou. U suprotnom, oREADY=1, sistem je spreman za izvršenje algoritma.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY CONTROL IS PORT (
  iCLK, inRESET:        IN  std_logic;
  iSTART, iY0, iCNT_0:  IN  std_logic;
  oLOAD_X, oLOAD_CNT,
  oLOAD_A, onRST_A,
  oLOAD_Y, oCE, oREADY: OUT std_logic );
END CONTROL;
```

```
ARCHITECTURE ARH_CONTROL OF CONTROL IS
  -- tip podataka za identifikaciju stanja automata
  TYPE tSTANJA IS (IDLE, INIT, CHECK_Y0, ADD, SHIFT, CHECK_CNT);
  -- deklaracija signala koji sadrze informaciju o
  -- tekucem i sledecem (narednom) stanju
  SIGNAL sSTANJE, sSLEDECE_STANJE : tSTANJA;
BEGIN

  -- kombinaciona mreza koja na osnovu trenutnog stanja
  -- i vrednosti ulaza generise kod narednog stanja
  PROCESS (iSTART, iY0, iCNT_0, sSTANJE) BEGIN
    CASE sSTANJE IS
      WHEN IDLE =>
        IF (iSTART = '0') THEN
          sSLEDECE_STANJE <= IDLE;
        ELSE
          sSLEDECE_STANJE <= INIT;
        END IF;
      WHEN INIT =>
        sSLEDECE_STANJE <= CHECK_Y0;
      WHEN CHECK_Y0 =>
        IF (iY0 = '0') THEN
          sSLEDECE_STANJE <= SHIFT;
        ELSE
          sSLEDECE_STANJE <= ADD;
        END IF;
      WHEN ADD =>
        sSLEDECE_STANJE <= SHIFT;
      WHEN SHIFT =>
        sSLEDECE_STANJE <= CHECK_CNT;
      WHEN CHECK_CNT =>
        IF (iCNT_0 = '0') THEN
          sSLEDECE_STANJE <= CHECK_Y0;
        ELSE
          sSLEDECE_STANJE <= IDLE;
        END IF;
    END CASE;
  END PROCESS;

  -- kombinaciona mreza za odredjivanje vrednosti
  -- izlaznog vektora na osnovu trenutnog stanja
  PROCESS (sSTANJE) BEGIN
    CASE sSTANJE IS
      WHEN IDLE =>
        oLOAD_X   <= '0'; oLOAD_Y <= '0';
        oLOAD_A   <= '0'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '0';
        oREADY    <= '1';
      WHEN INIT =>
        oLOAD_X   <= '1'; oLOAD_Y <= '1';
        oLOAD_A   <= '0'; onRST_A <= '0';
        oLOAD_CNT <= '1'; oCE     <= '0';
        oREADY    <= '0';
      WHEN CHECK_Y0 =>
        oLOAD_X   <= '0'; oLOAD_Y <= '0';
        oLOAD_A   <= '0'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '0';
```

```
        oREADY    <= '0';
      WHEN ADD =>
        oLOAD_X   <= '0'; oLOAD_Y <= '0';
        oLOAD_A   <= '1'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '0';
        oREADY    <= '0';
      WHEN SHIFT =>
        oLOAD_X   <= '0'; oLOAD_Y <= '0';
        oLOAD_A   <= '0'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '1';
        oREADY    <= '0';
      WHEN CHECK_CNT =>
        oLOAD_X   <= '0'; oLOAD_Y <= '0';
        oLOAD_A   <= '0'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '0';
        oREADY    <= '0';
    END CASE;
  END PROCESS;

  -- flip-flopovi za smestanje koda
  -- trenutnog stanja; postavljanje pocetnog
  -- stanja je sinhrono sa signalom takta
  PROCESS (iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK='1') THEN
      IF (inRESET = '0') THEN -- sinhroni reset
        sSTANJE <= IDLE;
      ELSE
        sSTANJE <= sSLEDECE_STANJE;
      END IF;
    END IF;
  END PROCESS;
END ARH_CONTROL;
```

MODUL MNOŽAČA:

U ovom modulu se instanciraju svi prethodno opisani moduli i međusobno povezuju u skladu sa blok šemom digitalnog sistema za množenje celih neoznačenih brojeva, Slika 8.30.

Prvo je potrebno formirati pakovanje sa opisom sprega svih modula koji se instanciraju:

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

PACKAGE MNOZAC_PKG IS
  COMPONENT REG
    GENERIC (
      pWIDTH: integer := 4
    );
    PORT (
      iCLK, inCLR: IN  STD_LOGIC;
      iCE: IN  STD_LOGIC;
      iD:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
      oQ:  OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
    );
  END COMPONENT;
```

```
    COMPONENT SHIFT_REG
      GENERIC (
        pWIDTH: integer := 4
      );
      PORT (
        iCLK, inCLR: IN  STD_LOGIC;
        iDSR, iLOAD, iSE: IN  STD_LOGIC;
        iD: IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
        oQ: OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
      );
    END COMPONENT;

    COMPONENT BROJAC
      GENERIC (
        pWIDTH: integer := 4
      );
      PORT (
        iCLK    : IN  STD_LOGIC;
        inRESET,
        iEN, iP : IN  STD_LOGIC;
        iDATA   : IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
        oQ      : OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
        oTSE    : OUT STD_LOGIC
      );
    END COMPONENT;

    COMPONENT ADDER
      GENERIC (
        pWIDTH: integer := 4
      );
      PORT (
        iX, iY:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
        oZ:      OUT STD_LOGIC_VECTOR(pWIDTH   DOWNTO 0)
      );
    END COMPONENT;

    COMPONENT CONTROL
      PORT (
        iCLK, inRESET:        IN  std_logic;
        iSTART, iY0, iCNT_0:  IN  std_logic;
        oLOAD_X, oLOAD_CNT,
        oLOAD_A, onRST_A,
        oLOAD_Y, oCE, oREADY: OUT std_logic
      );
    END COMPONENT;
  END MNOZAC_PKG;
```

Nakon formiranja pakovanja sa opisom sprega svih komponenti koje se instanciraju prelazi se na opis entiteta projektovanog digitalnog sistema. Projektovani modul množača sadrži sledeće ulazno/izlazne signale:

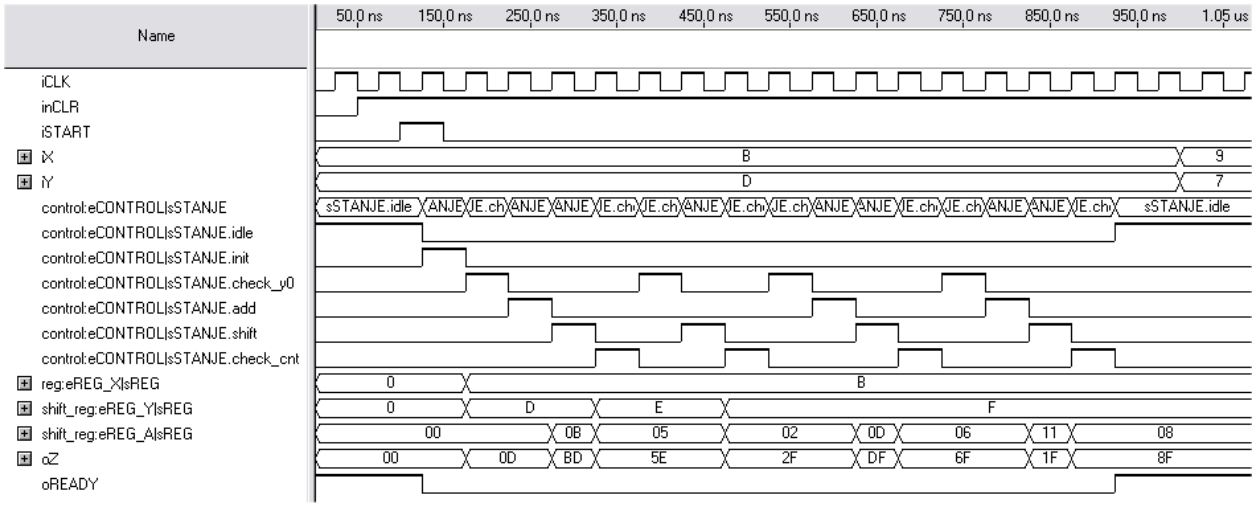
- `iCLK` takt signal,
- `inCLR` signal za brisanje sadržaja svih registara i brojača, aktivan na niskom nivou i sinhron sa signalom takta,
- `iSTART` impuls za početak izvršenja algoritma množenja,

- iX množenik, četvorobitni ulazni vektor,
- iY množitelj, četvorobitni ulazni vektor,
- oZ proizvod, četvorobitni izlazni vektor i
- oREADY izlazni signal koji govori da je u toku izvršenje algoritma, ako je njegova vrednost na niskom logičkom nivou. U suprotnom, oREADY=1, sistem je spreman za izvršenje algoritma.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE work.MNOZAC_PKG.all; -- ukljucivanje pakovanja sa komponentama

ENTITY MNOZAC IS
  PORT (
    iCLK, inCLR: IN  STD_LOGIC;
    iSTART: IN  STD_LOGIC;
    iX, iY: IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    oZ:     OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
    oREADY: OUT STD_LOGIC
  );
END MNOZAC;

ARCHITECTURE ARH_MNOZAC OF MNOZAC IS
  -- konsanta nula
  CONSTANT cZERO: STD_LOGIC := '0';
  -- broj bita sa kojima se reprezentuju operandi
  CONSTANT cNUM_BITS: STD_LOGIC_VECTOR(2 DOWNTO 0) := "100";

  -- registar mnozenika i mnozitelja
  SIGNAL sX, sY: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- akumulator i rezultat sabiraca
  SIGNAL sA, sR: STD_LOGIC_VECTOR(4 DOWNTO 0);
  -- brojac bita
  SIGNAL sCNT: STD_LOGIC_VECTOR(2 DOWNTO 0);
  -- upravljacki signali za registre i brojac
  SIGNAL sLOAD_X, sLOAD_Y, sLOAD_A,
         sLOAD_CNT, snRST_A, sCE,
         sCNT_TC, snCLR_A: STD_LOGIC;
BEGIN
  -- registar za smestanje mnozenika
  eREG_X: REG
    GENERIC MAP (pWIDTH=>4)
    PORT    MAP (iCLK=>iCLK, inCLR=>inCLR, iCE=>sLOAD_X,
                 iD=>iX, oQ=>sX);

  -- registar za smestanje mnozitelja
  eREG_Y: SHIFT_REG
    GENERIC MAP (pWIDTH=>4)
    PORT    MAP (iCLK=>iCLK, inCLR=>inCLR,
                 iDSR=>sA(0), iLOAD=>sLOAD_Y, iSE=>sCE,
                 iD=>iY, oQ=>sY);

  -- akumulator za smestanje rezultata i izlaznog prenosa
  snCLR_A <= inCLR AND snRST_A; -- resetovanje akumulatora
  eREG_A: SHIFT_REG
```

```
    GENERIC MAP (pWIDTH=>5)
    PORT    MAP (iCLK=>iCLK, inCLR=>snCLR_A,
                 iDSR=>cZERO, iLOAD=>sLOAD_A, iSE=>sCE,
                 iD=>sR, oQ=>sA);

  -- sabirac mnozenika i akumulisane vrednosti u akumulatoru
  -- rezultat se smesta nazad u akumulator
  eADDER: ADDER
    GENERIC MAP (pWIDTH=>4)
    PORT MAP (iX=>sX, iY=>sA(3 DOWNTO 0), oZ=>sR);

  -- brojac ciklusa (broja obradjenih bita) tokom mnozenja
  eCNT: BROJAC
    GENERIC MAP (pWIDTH=>3)
    PORT    MAP (iCLK=>iCLK, inRESET=>inCLR,
                 iEN=>sCE, iP=>sLOAD_CNT,
                 iDATA=>cNUM_BITS, oQ=>sCNT, oTSE=>sCNT_TC);

  -- automat za generisanje upravljackih signala
  eCONTROL: CONTROL
    PORT MAP (iCLK=>iCLK, inRESET=>inCLR,
              iSTART=>iSTART, iY0=>sY(0), iCNT_0=>sCNT_TC,
              oLOAD_X=>sLOAD_X, oLOAD_Y=>sLOAD_Y, oLOAD_A=>sLOAD_A,
              onRST_A=>snRST_A, oLOAD_CNT=>sLOAD_CNT,
              oCE=>sCE, oREADY=>oREADY);

  oZ <= (sA(3 DOWNTO 0) & sY); -- proizvod

END ARH_MNOZAC;
```

U ovoj implementaciji se može primetiti jedna razlika u odnosu na prikazanu blok šemu (Slika 8.30). Naime, u VHDL kodu množača nema zasebne instance jednobitnog registra C za smeštanje izlaznog prenosa sabirača. Ovaj registar je u ovom slučaju spojen sa akumulatorom, tako da akumulator ima umesto 4 ukupno 5 bita, gde je na mesto bita najveće važnosti postavljen izlazni prenos sabirača dok preostala 4 bita predstavljaju sadržaj akumulatora.



Slika 8.32: Vremenski dijagram rada realizovanog množača

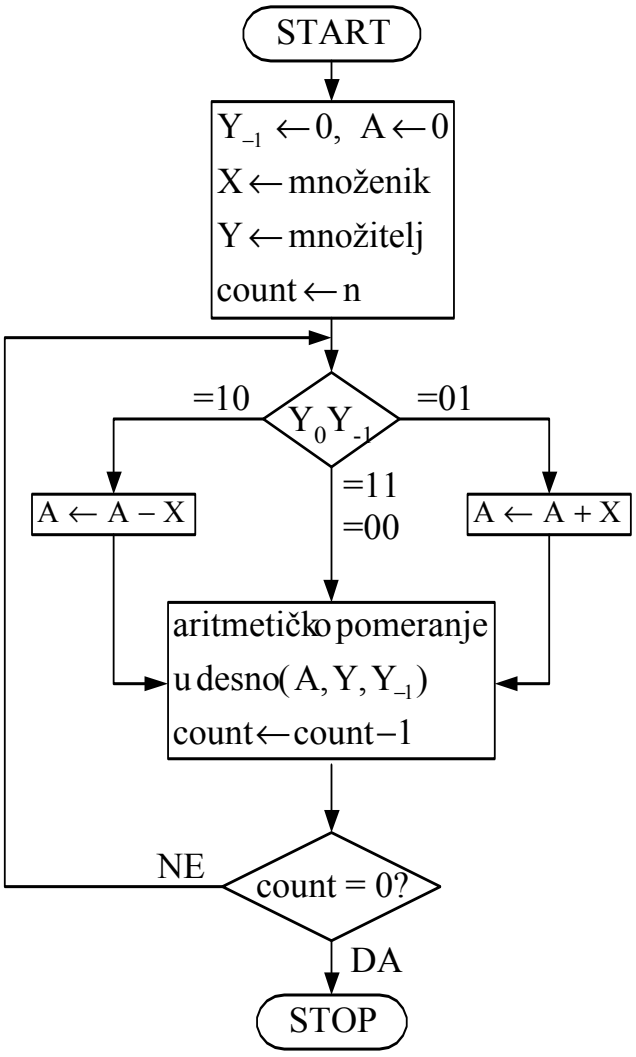
Slika 8.32 prikazuje vremenski dijagram izvršenja operacije množenja brojeva 11 i 13. Sve vrednosti na vremenskom dijagramu su u heksadecimalnom obliku, radi lakšeg poređenja sa vrednostima koje su prikazane prilikom ilustracije rada algoritma.

Uloga izlaznog signala oREADY je na osnovu vremenskog dijagrama očigledna. Naime, izvršenje operacije množenja je završeno kada signal dobije vrednost 1, i tada se može preuzeti dobijeni proizvod sa izlaznog vektora oZ.

8.9 ZADATAK:

Izvršiti sintezu digitalnog sistema za množenje celih označenih brojeva prema *Booth*-ovom algoritmu koji prikazuje Slika 8.29. Digitalni sistem treba projektovati za potrebe množenja četvorobitnih brojeva.

U rešenju treba prikazati blok šemu digitalnog sistema, graf prelazaka stanja generatora upravljačkih signala kao i VHDL kod kompletnog digitalnog sistema. Prikazati funkcionisanje isprojektovanog digitalnog sistema na primeru množenja brojeva 3 i 7, -3 i 7. 3 i –7 i na kraju –3 i -7.



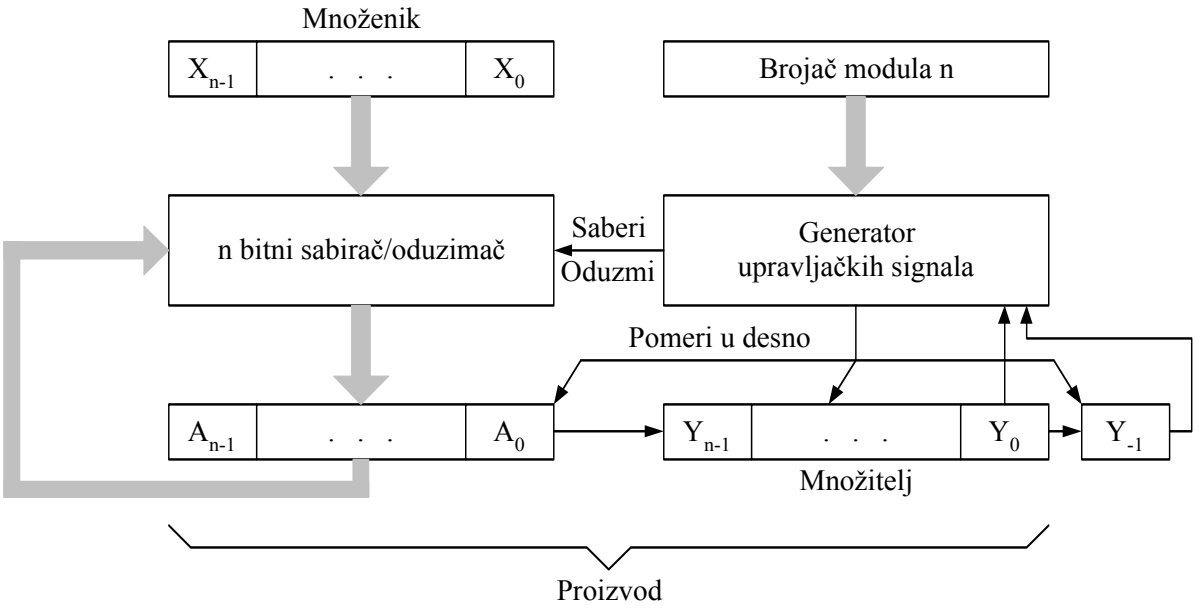
Slika 8.33: Blok dijagram algoritma za množenje celih označenih brojeva prema *Booth*-ovom algoritmu

REŠENJE:

Postoji više načina rešavanja problema množenja označenih brojeva. Jedan od njih je da se množenik i množitelj pretvore u pozitivne brojeve i izvrši se množenje pozitivnih brojeva po npr. algoritmu iz prethodnog zadatka. Na kraju se izvrši promena znaka proizvoda ako su znakovi množenika i množitelja različiti.

Međutim, prilikom implementacije množitelja češće se odabiraju metode kod kojih nije potrebna prethodno navedena operacija promena znaka rezultata. Jedno od rešenja za realizaciju množača celih označenih brojeva predstavlja realizacija prema *Booth*-ovom algoritmu.

Na osnovu blok dijagrama algoritma, Slika 8.33, vidi se da je za realizaciju traženog digitalnog sistema potrebno dva registra za smeštanje množenika X i množitelja Y. Takođe, potreban je i jednobitni registar Y_{-1} koji se smešta desno od bita najniže važnosti množitelja Y_0. Vrednost ovog registra se koristi prilikom određivanja operacije (sabiranje ili oduzimanje) koja će se izvršiti. Rezultat sabirača se smešta u akumulatorski registar A. Rezultat množenja će na kraju izvršenja algoritma biti smešten u registrima A i Y. Radi sinhronizacije realizacije algoritma potreban je jedan brojač i automat za generisanje upravljačkih signala, isto kao i u prethodnom zadatku.



Slika 8.34: Blok šema digitalnog sistema za množenje celih označenih brojava prema *Booth*-ovom algoritmu

Tokom izvršenja algoritma analiziraju se vrednosti bita množitelja krećući od bita najniže važnosti Y_0. Prilikom ove analize istovremeno se proverava i vrednost jednobitnog registra Y_{-1}. U slučaju da su ova dva bita različita izvršava se operacija sabiranja ili oduzimanja, zavisno od toga koji bit ima vrednost jedan. Nakon izvršene operacije sabiranja (oduzimanja) izvršava se aritmetičko pomeranje u desno sadržaja registra A, Y i Y_{-1}. Isto pomeranje se odmah izvršava u slučaju da biti Y_0 i Y_{-1} imaju istu vrednost. Aritmetičko pomeranje se izvršava tako da bit najveće važnosti akumulatora A, bit A_{n-1}, ostaje na svojoj poziciji i pomera se na poziciju susednog bita A_{n-2}. Na ovaj način dolazi do očuvanja znaka proizvoda koji se smešta u registre A i Y. Blok šemu digitalnog sistema za izvršenje *Booth*-ovog algoritma prikazuje Slika 8.34.

Procedura množenja prema ovom algoritmu se može ilustrovati na primeru množenja brojeva 7 i 3. Množenik ima vrednost $7_{DEC}=0411_{BIN}$, dok je množitelj $3_{DEC}=0011_{BIN}$. Njihov proizvod je $21_{DEC}=00010101_{BIN}$. Prema tome, na kraju izvršenja algoritma u registru A treba da je vrednost 0001_{BIN}, a u registru Y vrednost 0101_{BIN}. Za date vrednosti množenika i množitelja algoritam funcioniše na sledeći način:

A	Y	Y_{-1}	X		
0000	0011	0	0111	Inicijalne vrednosti	
1001	0011	0	0111	Oduzimanje	Prvi
1100	1001	1	0111	Pomeranje	ciklus
1110	0100	1	0111	Pomeranje	Drugi ciklus
0101	0100	1	0111	Sabiranje	Treći
0010	1010	0	0111	Pomeranje	ciklus
0001	0101	0	0111	Pomeranje	Četvrti ciklus

Za realizaciju traženog digitalnog sistema mogu se iskoristiti komponente brojača (`BROJAC`), registra (`REG`) i pomeračkog registra (`SHIFT_REG`) realizovane u prethodnom zadatku. Vrednost množenika X se smešta u realizovani registar označen entitetom `REG`, dok se vrednost množitelja Y smešta u pomerački registar `SHIFT_REG` zajedno sa jednobitnim registrom Y_{-1}. Na mesto bita najveće važnosti registra Y se dovodi bit najmanje važnosti akumulatorskog registra A. Akumulatorski registar je potrebno realizovati tako da obezbedi izvršenje operacije aritmetičkog pomeranja u desno, radi očuvanja znaka proizvoda. Zbog toga je potrebno realizovati modul registra sa mogućnošću aritmetičkog pomeranja u desno.

POMERAČKI REGISTAR (ARITMETIČKO POMERANJE):

Spregu ovog modula čine sledeći signali:

- `iCLK` takt signal,
- `inCLR` signal za brisanje sadržaja registra, aktivan na niskom nivou i sinhron sa signalom takta,
- `iLOAD` dozvola upisa u registar (ako je ovaj signal na visokom nivou u registar se na rastuću ivicu takt signala upisuje vrednost prisutna na ulaznom vektoru podataka `iD`),
- `iSE` dozvola pomeranja sadržaja registra u desno aktivna na visokom nivou,
- `iD` ulazni vektor podataka i
- `oQ` stanje registra,

Broj bita instanciranog registra se može odrediti preko generičkog parametra pWIDTH. Pretpostavlja se da instancirani registar ima četiri bita.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
ENTITY ASHIFT_REG IS
  GENERIC (
    pWIDTH: integer := 4    -- pretpostavljeni broj bita je 4
  );
  PORT (
    iCLK, inCLR: IN  STD_LOGIC;
    iLOAD, iSE:  IN  STD_LOGIC;
    iD: IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oQ: OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
  );
END ASHIFT_REG;

ARCHITECTURE ARH_ASHIFT_REG OF ASHIFT_REG IS
  -- stanje registra
  SIGNAL sREG: STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
BEGIN
  PROCESS (iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      -- sinhrono postavljanje pocetne vrednosti
      IF (inCLR = '0') THEN
        sREG <= (OTHERS => '0');
      ELSE
        IF (iLOAD = '1') THEN -- upis u registar
          sREG <= iD;
        ELSE
          IF (iSE = '1') THEN -- pomeranje u desno
            -- ARITMETICKO POMERANJE (II komplement)
            -- sREG(pWIDTH-1) sadrzi znak i taj bit se ne menja
            -- znak se povlaci za jedno mesto u desno
            -- ostali biti se pomeraju za jednu poziciju u desno
            FOR i IN 0 TO (pWIDTH-2) LOOP
              sREG(i) <= sREG(i+1);
            END LOOP;
          ELSE                -- zadrzavanje stanja
            sREG <= sREG;
          END IF;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  oQ <= sREG;   -- preslikavanje stanja registra na izlazni vektor

END ARH_ASHIFT_REG;
```

Pošto je sintetizovan pomerački registar sa generičkim parametrom koji određuje broj bita, operacija pomeranja je realizovana upotrebom FOR petlje:

```
FOR i IN 0 TO (pWIDTH-2) LOOP
  sREG(i) <= sREG(i+1);
END LOOP;
```

Pri tome se kreće od bita najniže važnosti (bit sa indeksom `0`) koji prima vrednost susednog bita veće važnosti, i tako redom sve do pretposlednjeg bita (bit sa indeksom `pWIDTH-2`) koji prima vrednost bita najveće važnosti (bit sa indeksom `pWIDTH-1`). Vrednost bita najveće važnosti se ne menja.

SABIRAČ:

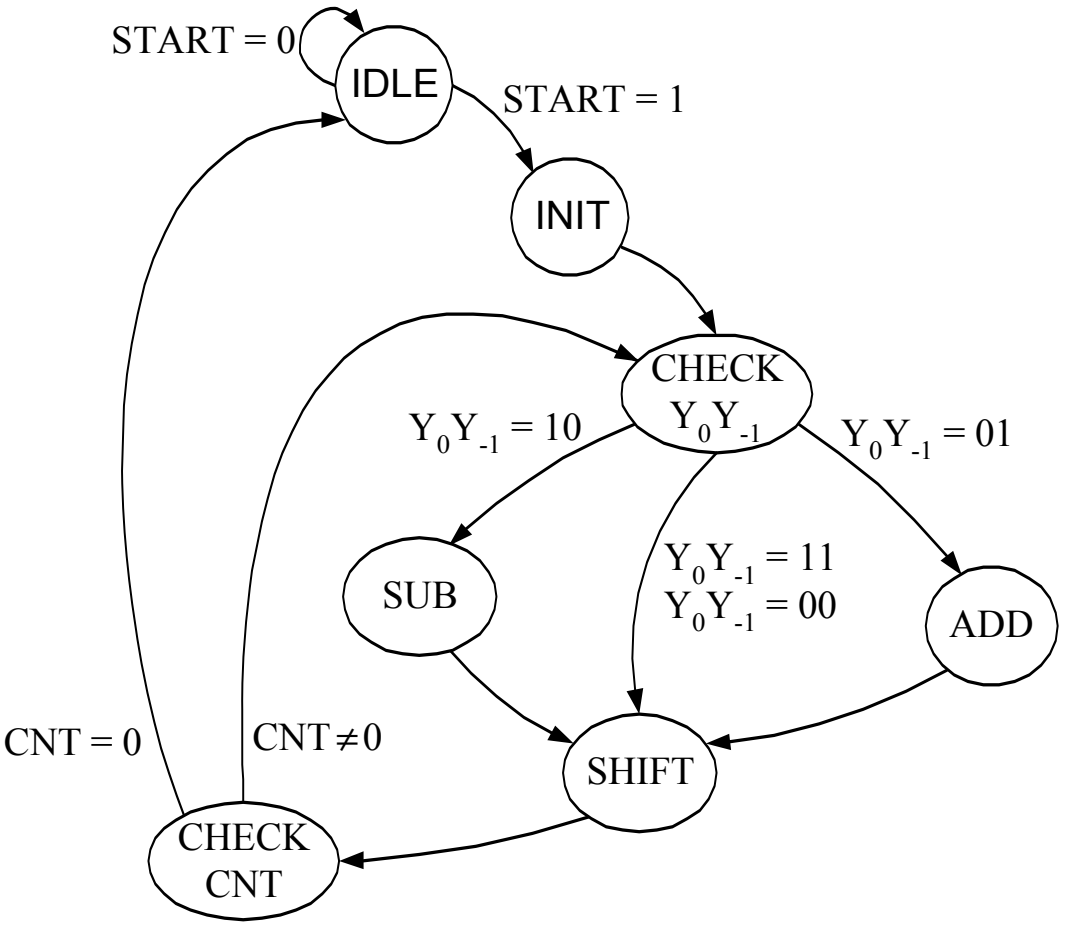
Modul sabirača u ovom slučaju mora da obezbedi izvršenje operacije sabiranja i oduzimanja. Zbog toga je potreban dodatni ulazni signal `iSEL` koji određuje koja operacija se izvršava. Ako je vrednost ovog ulaznog signala jednaka nuli tada se izvršava operacija sabiranja, dok se u suprotnom slučaju izvršava operacija oduzimanja ulaznih vektora, `iX` i `iY`. Ulazni vektori su tipa `STD_LOGIC_VECTOR` širine od `pWIDTH` bita. Izlazni vektor `oZ` je takođe širine od `pWIDTH` bita. `pWIDTH` je generički parametar koji određuje broj bita ulaznih i izlaznih vektora.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;

ENTITY ADDER IS
  GENERIC (
    pWIDTH: integer := 4    -- pretpostavljeni broj bita je 4
  );
  PORT (
    iX, iY:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    iSEL:    IN  STD_LOGIC;
    oZ:      OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
  );
END ADDER;

ARCHITECTURE ARH_ADDER OF ADDER IS
  SIGNAL sX, sY:     SIGNED(pWIDTH-1 DOWNTO 0);
  SIGNAL sADD, sSUB: SIGNED(pWIDTH-1 DOWNTO 0);
BEGIN
  -- konvertovanje ulaznih vektora u tip SIGNED
  sX <= SIGNED(iX);
  sY <= SIGNED(iY);
  sADD <= sX + sY; -- sabiranje
  sSUB <= sX - sY; -- oduzimanje

  -- konvertovanje rezultata u tip STD_LOGIC_VECTOR
  oZ <= STD_LOGIC_VECTOR(sADD) WHEN (iSEL = '0') ELSE
        STD_LOGIC_VECTOR(sSUB);
END ARH_ADDER;
```

U ovom slučaju se ulazni vektori konvertuju u tip `SIGNED` pre same realizacije sabirača. Vrednosti izlaznog prenosa (pozajmice) se ignoriše (tj. nije realizovano) pošto se ne koriste prilikom izvršenja algoritma Rezultat sabiranja i oduzimanja se konvertuje u tip `STD_LOGIC_VECTOR` i prosleđuje na izlazni vektor `oZ` u zavisnosti od vrednosti ulaznog signala `iSEL`.

MODUL ZA GENERISANJE UPRAVLJAČKIH SIGNALA:

Ovaj modul treba da na osnovu zadatog blok dijagram algoritma množenja označenih brojeva prema *Booth*-ovom algoritmu, Slika 8.33, generiše odgovarajuće upravljačke signale za registre, brojač i sabirač. I u ovom slučaju će se ovaj modul realizovati pomoću Murovog automata. Radi lakše implementacije formira se graf prelazaka stanja, Slika 8.35, gde se u svakom stanju generišu odgovarajući upravljački signali u skladu sa potrebnom operacijom u tom vremenskom trenutku.



Slika 8.35: Graf prelazaka stanja upravljačkog automata

Automat se inicijalno nalazi u stanju `IDLE`. U slučaju da se na ulazu pojavi impuls za početak izvršenja algoritma, `START=1`, automat prelazi u stanje `INIT` gde se izvršava inicijalizacija stanja svih registara i brojača. Nakon toga prelazi se u stanje `CHECK_Y10` gde se proverava vrednost prva dva bita registra koji sadrži vrednost množitelja i vrednost bita Y_{-1}. Ovde se zapravo proveravaju vrednosti bita najniže važnosti množitelja Y_0 i vrednost jednobitnog registra Y_{-1}. U slučaju da su vrednosti ova dva bita različita, prelazi se u stanje `SUB` u slučaju da je $Y_0Y_{-1}=10$ ili u stanje `ADD` ako je $Y_0Y_{-1}=01$. Ako dva navedena bita imaju iste vrednosti prelazi se u stanje `SHIFT`, gde se takođe bezuslovno prelazi i iz stanja `SUB` i `ADD`. U stanju `SHIFT` se izvršava aritmetičko pomeranje registara `A`, `Y` i Y_{-1}, kao i

dekrementiranje vrednosti brojača. U sledećem stanju, CHECK_CNT, proverava se da li je brojač stigao do kraja ciklusa brojanja, kada se automat vraća u inicijalno stanje IDLE. Ako brojač nije stigao do kraja ciklusa brojanja prelazi se u stanje CHECK_Y10 i postupak se ponavlja do kraja ciklusa brojanja. Tada se završava izvršenje algoritma množenja neoznačenih celih brojeva.

Upravljački automat u ovom slučaju ima sledeće ulazne signale:

- iCLK takt signal,
- inRESET signal za postavljanje automata u inicijalno stanje, aktivan na niskom nivou i sinhron sa signalom takta,
- iSTART impuls za početak izvršenja algoritma množenja,
- iY10 vrednost dva bita najniže važnosti registra gde su smešteni množitelj i bit Y_{-1}, na osnovu koje se odlučuje da li će se izvršiti operacija sabiranja, oduzimanja ili se neće izvršiti nijedna operacija,
- iCNT_0 indikacija kraja ciklusa brojanja brojača ciklusa množenja, aktivna na visokom nivou.

Potrebno je generisati sledeće upravljačke signale:

- oLOAD_CNT impuls za upis početne vrednosti brojača,
- oLOAD_X impuls za upis množioca,
- oLOAD_Y impuls za upis množitelja,
- oLOAD_A impuls za upis rezultata sabiranja u akumulator,
- onRST_A impuls za brisanje sadržaja akumulatora, aktivan na niskom nivou,
- oCE dozvola pomeranja i dozvola brojanja,
- oADD_SUB upravljački signal za određivanje koja će se operacija izvršiti. Ako je ovaj signal na niskom nivou izvršiće se sabiranje, dok će se oduzimanje izvršiti u suprotnom slučaju,
- oREADY izlazni signal koji govori da je u toku izvršenje algoritma, ako je njegova vrednost na niskom logičkom nivou. U suprotnom, oREADY=1, sistem je spreman za izvršenje algoritma.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY CONTROL IS PORT (
  iCLK, inRESET, iSTART: IN  STD_LOGIC;
  iY10:   IN STD_LOGIC_VECTOR(1 DOWNTO 0);
  iCNT_0: IN STD_LOGIC;
  oLOAD_X, oLOAD_Y, oLOAD_A, oLOAD_CNT,
  oCE, onRST_A, oADD_SUB, oREADY: OUT STD_LOGIC );
END CONTROL;
```

```
ARCHITECTURE ARH_CONTROL OF CONTROL IS
  -- tip podataka za identifikaciju stanja automata
  TYPE tSTANJA IS
               (IDLE, INIT, CHECK_Y10, ADD, SUB, SHIFT, CHECK_CNT);
  -- deklaracija signala koji sadrze informaciju o
  -- tekucem i sledecem (narednom) stanju
  SIGNAL sSTANJE, sSLEDECE_STANJE : tSTANJA;
BEGIN
  -- kombinaciona mreza koja na osnovu trenutnog stanja
  -- i vrednosti ulaza generise kod narednog stanja
  PROCESS (iSTART, iY10, iCNT_0, sSTANJE) BEGIN
    CASE sSTANJE IS
      WHEN IDLE =>
        IF (iSTART = '0') THEN
          sSLEDECE_STANJE <= IDLE;
        ELSE
          sSLEDECE_STANJE <= INIT;
        END IF;
      WHEN INIT =>
        sSLEDECE_STANJE <= CHECK_Y10;
      WHEN CHECK_Y10 =>
        IF (iY10 = "01") THEN
          sSLEDECE_STANJE <= ADD;
        ELSIF (iY10 = "10") THEN
          sSLEDECE_STANJE <= SUB;
        ELSE
          sSLEDECE_STANJE <= SHIFT;
        END IF;
      WHEN ADD =>
        sSLEDECE_STANJE <= SHIFT;
      WHEN SUB =>
        sSLEDECE_STANJE <= SHIFT;
      WHEN SHIFT =>
        sSLEDECE_STANJE <= CHECK_CNT;
      WHEN CHECK_CNT =>
        IF (iCNT_0 = '0') THEN
          sSLEDECE_STANJE <= CHECK_Y10;
        ELSE
          sSLEDECE_STANJE <= IDLE;
        END IF;
    END CASE;
  END PROCESS;

  -- kombinaciona mreza za odredjivanje vrednosti
  -- izlaznog vektora na osnovu trenutnog stanja
  PROCESS (sSTANJE) BEGIN
    CASE sSTANJE IS
      WHEN IDLE =>
        oLOAD_X <= '0'; oLOAD_Y   <= '0';
        oLOAD_A <= '0'; oLOAD_CNT <= '0';
        oCE     <= '0'; oADD_SUB  <= '0';
        onRST_A <= '1'; oREADY    <= '1';
      WHEN INIT =>
        oLOAD_X <= '1'; oLOAD_Y   <= '1';
        oLOAD_A <= '0'; oLOAD_CNT <= '1';
        oCE     <= '0'; oADD_SUB  <= '0';
        onRST_A <= '0'; oREADY    <= '0';
```

```
        WHEN CHECK_Y10 =>
          oLOAD_X <= '0'; oLOAD_Y   <= '0';
          oLOAD_A <= '0'; oLOAD_CNT <= '0';
          oCE     <= '0'; oADD_SUB  <= '0';
          onRST_A <= '1'; oREADY    <= '0';
        WHEN ADD =>
          oLOAD_X <= '0'; oLOAD_Y   <= '0';
          oLOAD_A <= '1'; oLOAD_CNT <= '0';
          oCE     <= '0'; oADD_SUB  <= '0';
          onRST_A <= '1'; oREADY    <= '0';
        WHEN SUB =>
          oLOAD_X <= '0'; oLOAD_Y   <= '0';
          oLOAD_A <= '1'; oLOAD_CNT <= '0';
          oCE     <= '0'; oADD_SUB  <= '1';
          onRST_A <= '1'; oREADY    <= '0';
        WHEN SHIFT =>
          oLOAD_X <= '0'; oLOAD_Y   <= '0';
          oLOAD_A <= '0'; oLOAD_CNT <= '0';
          oCE     <= '1'; oADD_SUB  <= '0';
          onRST_A <= '1'; oREADY    <= '0';
        WHEN CHECK_CNT =>
          oLOAD_X <= '0'; oLOAD_Y   <= '0';
          oLOAD_A <= '0'; oLOAD_CNT <= '0';
          oCE     <= '0'; oADD_SUB  <= '0';
          onRST_A <= '1'; oREADY    <= '0';
      END CASE;
    END PROCESS;

    -- flip-flopovi za smestanje koda
    -- trenutnog stanja; postavljanje pocetnog
    -- stanja je sinhrono sa signalom takta
    PROCESS (iCLK) BEGIN
      IF (iCLK'EVENT AND iCLK='1') THEN
        IF (inRESET = '0') THEN -- sinhroni reset
          sSTANJE <= IDLE;
        ELSE
          sSTANJE <= sSLEDECE_STANJE;
        END IF;
      END IF;
    END PROCESS;

  END ARH_CONTROL;
```

MODUL MNOŽAČA:

U ovom modulu se instanciraju svi prethodno opisani moduli i međusobno povezuju u skladu sa blok šemom digitalnog sistema za množenje celih označenih brojeva, Slika 8.34.

Prvo je potrebno formirati pakovanje sa opisom sprega svih modula koji se instanciraju:

```
  LIBRARY ieee;
    USE ieee.std_logic_1164.all;

  PACKAGE MNOZAC_PKG IS
    COMPONENT REG
```

```
  GENERIC (
    pWIDTH: integer := 4
  );
  PORT (
    iCLK, inCLR: IN  STD_LOGIC;
    iCE: IN  STD_LOGIC;
    iD:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oQ:  OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
  );
END COMPONENT;

COMPONENT SHIFT_REG
  GENERIC (
    pWIDTH: integer := 4
  );
  PORT (
    iCLK, inCLR: IN  STD_LOGIC;
    iDSR, iLOAD, iSE: IN  STD_LOGIC;
    iD: IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oQ: OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
  );
END COMPONENT;

COMPONENT ASHIFT_REG
  GENERIC (
    pWIDTH: integer := 4
  );
  PORT (
    iCLK, inCLR: IN  STD_LOGIC;
    iLOAD, iSE:  IN  STD_LOGIC;
    iD: IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oQ: OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
  );
END COMPONENT;

COMPONENT BROJAC
  GENERIC (
    pWIDTH: integer := 4
  );
  PORT (
    iCLK    : IN  STD_LOGIC;
    inRESET,
    iEN, iP : IN  STD_LOGIC;
    iDATA   : IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oQ      : OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oTSE    : OUT STD_LOGIC
  );
END COMPONENT;

COMPONENT ADDER
  GENERIC (
    pWIDTH: integer := 4
  );
  PORT (
    iX, iY:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    iSEL:    IN  STD_LOGIC;
    oZ:      OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
```

```
    );
  END COMPONENT;

  COMPONENT CONTROL
    PORT (
      iCLK, inRESET, iSTART: IN  STD_LOGIC;
      iY10:   IN STD_LOGIC_VECTOR(1 DOWNTO 0);
      iCNT_0: IN STD_LOGIC;
      oLOAD_X, oLOAD_Y, oLOAD_A, oLOAD_CNT,
      oCE, onRST_A, oADD_SUB, oREADY: OUT STD_LOGIC
    );
  END COMPONENT;

END MNOZAC_PKG;
```

Nakon formiranja pakovanja sa opisom sprega svih komponenti koje se instanciraju prelazi se na opis entiteta projektovanog digitalnog sistema. Projektovani modul množača sadrži sledeće ulazno/izlazne signale:

- `iCLK` takt signal,
- `inCLR` signal za brisanje sadržaja svih registara i brojača, aktivan na niskom nivou i sinhron sa signalom takta,
- `iSTART` impuls za početak izvršenja algoritma množenja,
- `iX` množenik, četvorobitni ulazni vektor,
- `iY` množitelj, četvorobitni ulazni vektor,
- `oZ` proizvod, četvorobitni izlazni vektor i
- `oREADY` izlazni signal koji govori da je u toku izvršenje algoritma, ako je njegova vrednost na niskom logičkom nivou. U suprotnom, `oREADY=1`, sistem je spreman za izvršenje algoritma.

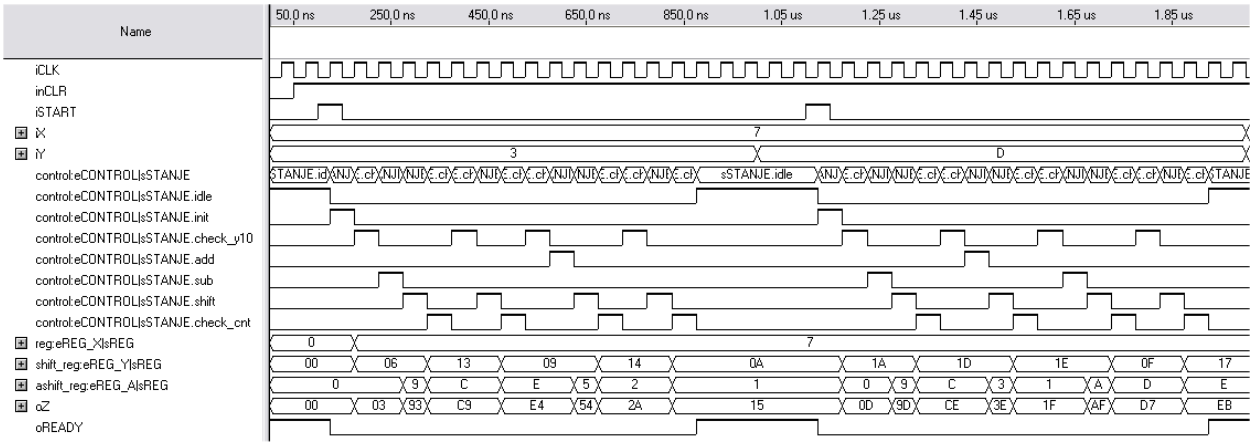
```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE work.MNOZAC_PKG.all; -- ukljucivanje pakovanja sa komponentama

ENTITY MNOZAC IS
  PORT (
    iCLK, inCLR: IN  STD_LOGIC;
    iSTART: IN  STD_LOGIC;
    iX, iY: IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    oZ:     OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
    oREADY: OUT STD_LOGIC
  );
END MNOZAC;

ARCHITECTURE ARH_MNOZAC OF MNOZAC IS
  -- broj bita sa kojima se reprezentuju operandi
  CONSTANT cNUM_BITS: STD_LOGIC_VECTOR(2 DOWNTO 0) := "100";

    -- registar mnozenika, akumulator i rezultat sabiraca/oduzimaca
  SIGNAL sX, sA, sR: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- registar mnozitelja prosiren sa donje strane dodatnim bitom
  SIGNAL sY: STD_LOGIC_VECTOR(4 DOWNTO 0);
```

```
    -- ulazni signal u registar mnozitelja
    SIGNAL sDATA: STD_LOGIC_VECTOR(4 DOWNTO 0);
    -- brojac bita
    SIGNAL sCNT: STD_LOGIC_VECTOR(2 DOWNTO 0);
    -- upravljacki signali za registre, brojac i sabirac/oduzimac
    SIGNAL sLOAD_X, sLOAD_Y, sLOAD_A, sLOAD_CNT,
           sCE, snCLR_A, snRST_A, sADD_SUB, sCNT_TC: STD_LOGIC;
  BEGIN
    -- registar za smestanje mnozenika
    eREG_X: REG
      GENERIC MAP (pWIDTH=>4)
      PORT    MAP (iCLK=>iCLK, inCLR=>inCLR, iCE=>sLOAD_X,
                   iD=>iX, oQ=>sX);

    -- registar za smestanje mnozitelja
    sDATA <= (iY & '0');  -- ulazna vrednost registra
    eREG_Y: SHIFT_REG
      GENERIC MAP (pWIDTH=>5)
      PORT    MAP (iCLK=>iCLK, inCLR=>inCLR,
                   iDSR=>sA(0), iLOAD=>sLOAD_Y, iSE=>sCE,
                   iD=>sDATA, oQ=>sY);

    -- akumulator
    snCLR_A <= inCLR AND snRST_A; -- signal za resetovanje akumulatora
    eREG_A: ASHIFT_REG
      GENERIC MAP (pWIDTH=>4)
      PORT    MAP (iCLK=>iCLK, inCLR=>snCLR_A,
                   iLOAD=>sLOAD_A, iSE=>sCE, iD=>sR, oQ=>sA);

    -- sabirac/oduzimac mnozenika i
    -- akumulisane vrednosti u akumulatoru
    -- rezultat se smesta nazad u akumulator
    eADDER: ADDER
      GENERIC MAP (pWIDTH=>4)
      PORT MAP (iX=>sA, iY=>sX, iSEL=>sADD_SUB, oZ=>sR);

    -- brojac ciklusa (broja obradjenih bita) tokom mnozenja
    eCNT: BROJAC
      GENERIC MAP (pWIDTH=>3)
      PORT    MAP (iCLK=>iCLK, inRESET=>inCLR, iEN=>sCE,
                   iP=>sLOAD_CNT, iDATA=>cNUM_BITS,
                   oQ=>sCNT, oTSE=>sCNT_TC);

    -- automat ze generisanje upravljackih signala
    eCONTROL: CONTROL
      PORT MAP (iCLK=>iCLK, inRESET=>inCLR,
                iSTART=>iSTART, iY10=>sY(1 DOWNTO 0), iCNT_0=>sCNT_TC,
                oLOAD_X=>sLOAD_X, oLOAD_Y=>sLOAD_Y, oLOAD_A=>sLOAD_A,
                oLOAD_CNT=>sLOAD_CNT, oCE=>sCE, onRST_A=>snRST_A,
                oADD_SUB=>sADD_SUB, oREADY=>oREADY);

    oZ <= (sA & sY(4 DOWNTO 1)); -- proizvod

  END ARH_MNOZAC;
```

Slično kao u prethodnom zadatku, i u ovom slučaju je jednobitni registar spojen sa susednim registrom. U ovom slučaju je jednobitni registar Y_{-1} spojen sa

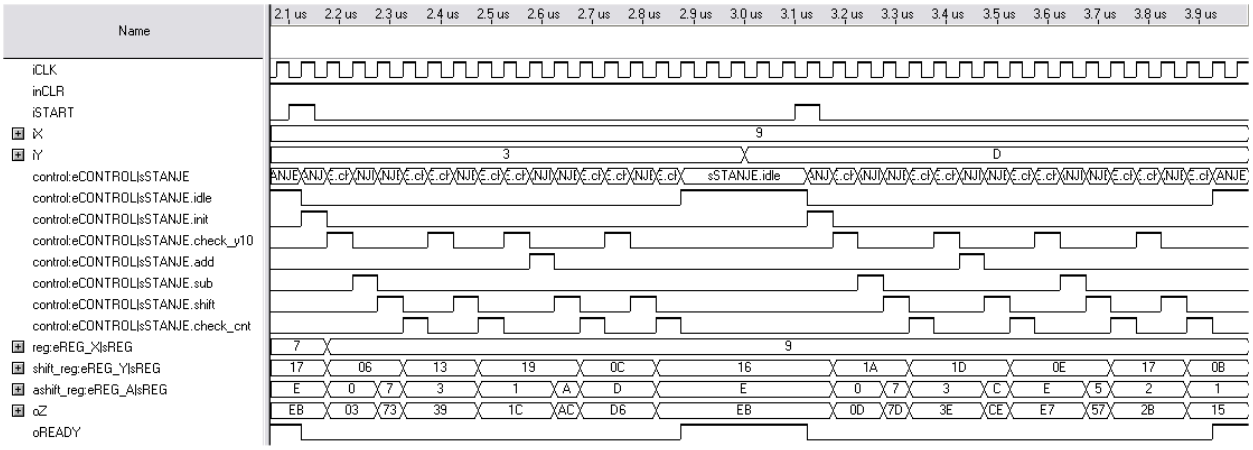
registrom za smeštanje množitelja. Time registar Y ima ukupno 5 bita, gde je na mestu bita najniže važnosti smeštena vrednost Y_{-1} dok preostala 4 bita predstavljaju množitelj Y. Zbog toga je uveden signal `sDATA` koji sadrži inicijalnu vrednost registra, tako da bit najniže važnosti prilikom inicijalizacije bude postavljen na nulu, a preostali biti primaju vrednost ulaznog vektora množitelja (`sDATA<=(iY & '0')`).

Slika 8.36 prikazuje vremenski dijagram izvršenja operacije množenja broja 7 sa 3 i sa –3. Sve vrednosti na vremenskom dijagramu su prikazane u heksadecimalnom obliku. Proizvod 7·3=21 je u heksadecimalnom obliku 15, dok proizvod –3·7=–21 se heksadecimalno predstavlja sa EB, kao što se i može videti na vremenskom dijagramu.



Slika 8.36: Simulacija rada množača za množenje 7 sa 3 i 7 sa –3

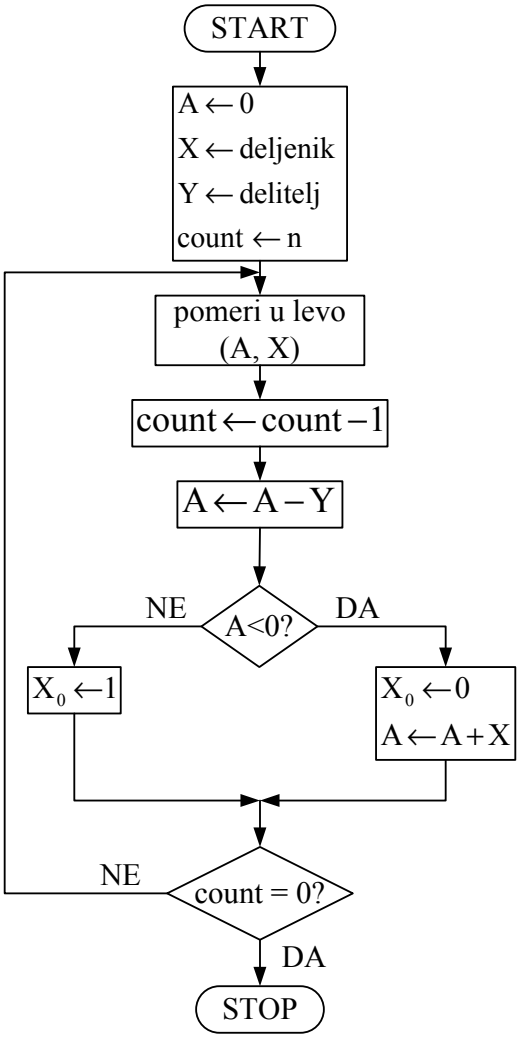
Vremenski dijagram izvršenja operacije množenja broja –7 sa 3 i sa –3 prikazuje Slika 8.37, gde su takođe sve vrednosti prikazane u heksadecimalnom obliku. Proizvod –7·3=–21 je u heksadecimalnom obliku EB, dok proizvod –3·(–7)=21 se heksadecimalno predstavlja sa 15, kao što se i može videti na vremenskom dijagramu.



Slika 8.37: Simulacija rada množača za množenje –7 sa 3 i 7 sa –3

8.10 ZADATAK:

Izvršiti sintezu digitalnog sistema za delenje celih neoznačenih brojeva prema algoritmu koji prikazuje Slika 8.38. Digitalni sistem treba projektovati za potrebe delenja četvorobitnih brojeva.



Slika 8.38: Blok dijagram algoritma za delenje celih neoznačenih brojeva

U rešenju treba prikazati blok šemu digitalnog sistema, graf prelazaka stanja generatora upravljačkih signala kao i VHDL kod kompletnog digitalnog sistema.

REŠENJE:

Delenje je kompleksnija operacija u odnosu na množenje. Bazira se takođe na ručnom algoritmu delenja i svodi se na uzastopno izvršenje operacija pomeranja, sabiranja ili oduzimanja. Zbog toga se za projektovanje aritmetičke mreže za delenje brojeva mogu primeniti isti principi kao i za projektovanje aritmetičkih mreža za množenje iz prethodna dva zadatka.

Pre same operacije deljenja potrebno je utvrditi da li je deljenik veći od delitelja, čime se utvrđuje da li delitelj može da deli deljenik. Ukoliko se delitelj predstavlja sa manjim brojem bita proširuje se sa nulama radi izjednačavanja broja

bita delitelja i deljenika. Nakon toga se izvšava oduzimanje delitelja od prve parcijalne vrednosti deljenika s leve strane koja je veća od delitelja. Dobijena vrednost predstavlja delimični ostatak. U sledećem koraku se vrši oduzimanje od vrednosti delimičnog ostatka spojene sa prvim sledećim bitom deljenika, čime se dobija novi delimični ostatak. Postupak se ponavlja za sve bite deljenika.

Opisani algoritam ilustruje sledeći primer:

delitelj: Y = 1011 = (11_{10}).
deljenik: X = 10010011 = (147_{10})

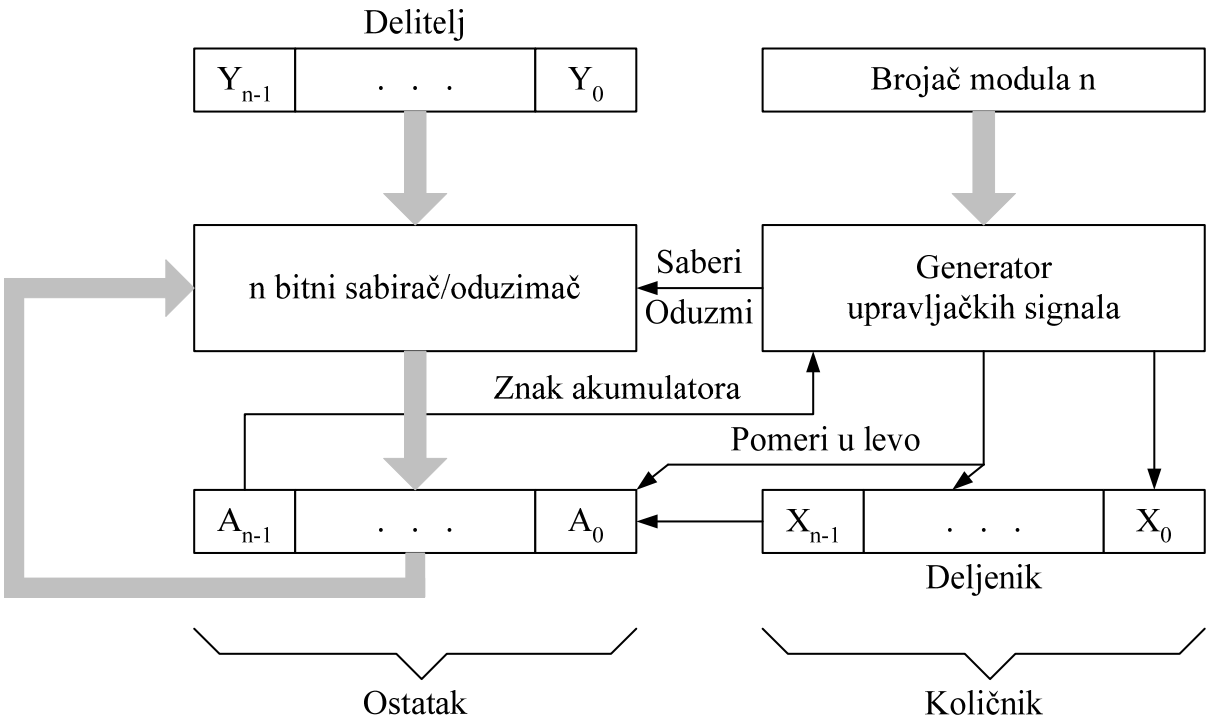
Proces dobijanja količnika, korak po korak je sledeći:

deljenik X									delitelj Y				količnik Z				
1	0	0	1	0	0	1	1	:	1	0	1	1	=	1	1	0	1
-	1	0	1	1	↓												
		1	1	1	0				delimični ostatak: 111								
	-	1	0	1	1	↓	↓										
				1	1	1	1		delimični ostatak: 11								
			-	1	0	1	1										
					1	0	0		ostatak								

Dobijena vrednost količnika je $1101=13_{10}$, dok je ostatak $100=4_{10}$ $(13\cdot11+4=147)$.

Opisani postupak prikazuje dati blok dijagram algoritma (Slika 8.38) sa modifikacijom da se delimični ostatak pomera ulevo u odnosu na fiksni delitelj. Potrebno je napomenuti da je za uspešno izvršenje algoritma potrebno obezbediti da deljenik X bude veći od delitelja Y.

Slika 8.39 prikazuje blok šemu digitalnog sistema za izvršenje traženog algoritma. Delitelj se smešta u registar Y i sadržaj ovog registra se nemenja u toku izvršenja algoritma. Deljenik se smešta u registar X čiji se sadržaj pomera, zajedno sa akumulatorskim registrom A, ulevo za jednu poziciju sa svakim ciklusom algoritma. Na bit najniže važnosti registra X, označen sa X_0, se postavljaju dobijeni biti količnika. Delitelj Y se oduzima od sadržaja akumulatorskog registra A da bi se odredilo da li Y deli delimični ostatak smešten u A. U tom slučaju je $A>0$ i X_0 se postavlja na vrednost 1. Inače se X_0 postavlja na vrednost 0 i deljenik se dodaje na sadržaj akumulatorskog registra radi obnavljanja prethodne vrednosti delimičnog ostatka. Nakon toga se smanjuje vrednost brojača i ceo postupak se ponavlja za sve bite deljenika. Na kraju izvršenja algoritma količnik se nalazi u registru X, dok akumulatorski registar A sadrži ostatak deljenja.



Slika 8.39: Blok šema digitalnog sistema za delenje celih neoznačenih brojeva

Za sintezu traženog digitalnog sistema, pored globalnog modula koji povezuje sve komponente, potrebno je isprojektovati pomerački registar koji pomera svoj sadržaj ulevo i automat koji će generisati odgovarajuće upravljačke signale. VHDL opis sabirača, brojača i registra za smeštanje delitelja Y je identičan kao i u prethodnom zadatku.

POMERAČKI REGISTAR:

Modul sa opisom pomeračkog registra ima generički parametar `pWIDTH` koji definiše broj bita registra koji se instancira. Pretpostavlja se da instancirani pomerački registar ima četiri bita. Smer pomeranja je u levo.

Spregu modula pomeračkog registra čine sledeći signali:

- `iCLK` takt signal,
- `inCLR` signal za brisanje sadržaja registra, aktivan na niskom nivou i sinhron sa signalom takta,
- `iDSL` *Data Shift Left*, vrednost koja se upisuje u bit najmanje važnosti registra prilikom pomeranja u levo,
- `iLOAD` dozvola upisa u registar (ako je ovaj signal na visokom nivou u registar se na rastuću ivicu takt signala upisuje vrednost prisutna na ulaznom vektoru podataka `iD`),
- `iSE` dozvola pomeranja sadržaja registra u levo aktivna na visokom nivou,
- `iD` ulazni vektor podataka i
- `oQ` stanje registra,

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY SHIFT_REG IS
  GENERIC (
    pWIDTH: integer := 4    -- pretpostavljeni broj bita je 4
  );
  PORT (
    iCLK, inCLR: IN  STD_LOGIC;
    iLOAD, iSE, iDSL: IN  STD_LOGIC;
    iD: IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oQ: OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
  );
END SHIFT_REG;

ARCHITECTURE ARH_SHIFT_REG OF SHIFT_REG IS
  -- stanje registra
  SIGNAL sREG: STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
BEGIN

  PROCESS (iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      -- sinhrono postavljanje pocetne vrednosti
      IF (inCLR = '0') THEN
        sREG <= (OTHERS => '0');
      ELSE
        IF (iLOAD = '1') THEN -- upis u registar
          sREG <= iD;
        ELSE
          IF (iSE = '1') THEN -- pomeranje u levo
            sREG(0) <= iDSL;
            FOR i IN (pWIDTH-1) DOWNTO 1 LOOP
              sREG(i) <= sREG(i-1);
            END LOOP;
          ELSE                  -- zadrzavanje stanja
            sREG <= sREG;
          END IF;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- preslikavanje stanja registra na izlazni vektor
  oQ <= sREG;

END ARH_SHIFT_REG;
```

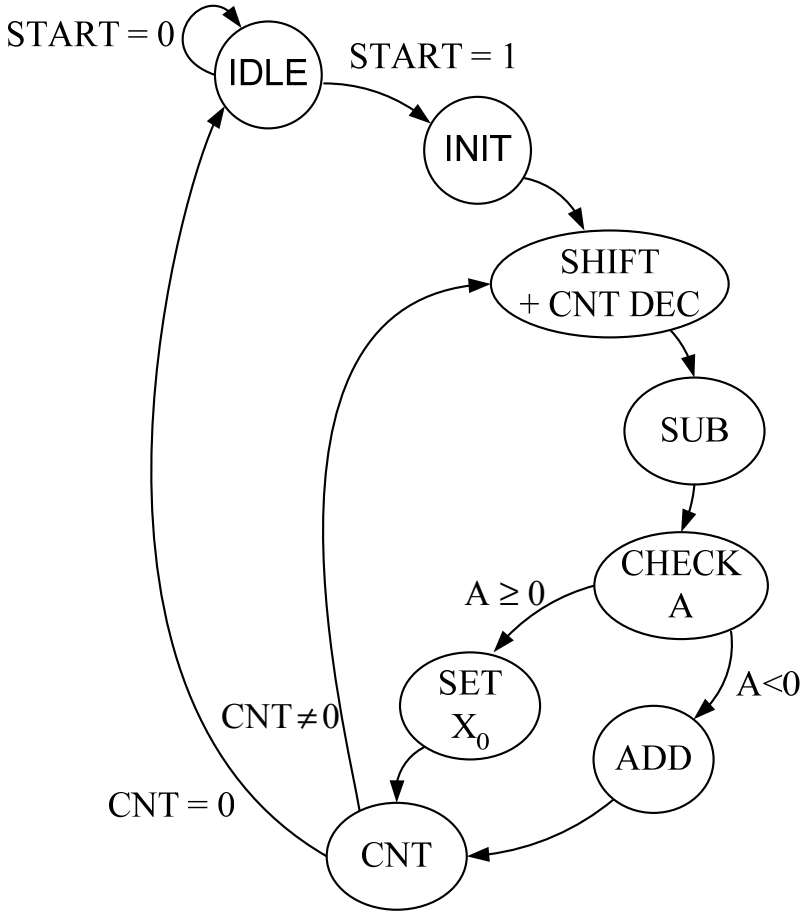
Pošto je sintetizovan pomerački registar sa generičkim parametrom koji određuje broj bita, operacija pomeranja je realizovana upotrebom FOR petlje:

```
            sREG(0) <= iDSL;
            FOR i IN (pWIDTH-1) DOWNTO 1 LOOP
              sREG(i) <= sREG(i-1);
            END LOOP;
```

Pri tome se kreće od bita najveće važnosti (bit sa indeksom `pWIDTH-1`) koji prima vrednost susednog bita manje važnosti, i tako redom sve do bita sa indeksom `1`. Stanje na ulaznom signalu `iDSL` se upisuje u bit najmanje važnosti (bit sa indeksom `0`).

MODUL ZA GENERISANJE UPRAVLJAČKIH SIGNALA:

Ovaj modul treba da na osnovu zadatog blok dijagram algoritma delenja neoznačenih celih brojeva, Slika 8.38, generiše odgovarajuće upravljačke signale za registre, brojač i sabirač. Modul za generisanje upravljačkih signala se realizuje pomoću Murovog automata. Radi lakše implementacije formira se graf prelazaka stanja, Slika 8.40, gde se u svakom stanju generišu odgovarajući upravljački signali u skladu sa potrebnom operacijom u tom vremenskom trenutku.



Slika 8.40: Graf prelazaka stanja upravljačkog automata

Automat se inicijalno nalazi u stanju `IDLE`. U slučaju da se na ulazu pojavi impuls za početak izvršenja algoritma, `START=1`, automat prelazi u stanje `INIT` gde se izvršava inicijalizacija stanja svih registara i brojača. Nakon toga prelazi se u stanje `SHIFT` gde se izvršava pomeranje sadržaja registara A i Y. Takođe, u ovom stanju se i dekrementuje sadržaj brojača. To je mala modifikacija datog algoritma, ali ona ne utiče na njegovo izvršenje pošto se vrednost brojača

proverava tek na kraju ciklusa. Modifikacija je urađena radi zadržavanja jedinstvenog upravljačkog signala za menjanje sadržaja pomeračkih registara i brojača, kao i kod realizacije množača iz prethodna dva zadatka. Bez ove modifikacije bilo bi potrebno uvesti dva nezavisna upravljačka signala za dozvolu pomeranja pomeračkih registara i za dozvolu brojanja brojača.

Iz stanja `SHIFT` se prelazi u stanje `SUB` gde se generišu upravljački signali za oduzimanje delitelja Y od delimičnog ostatka smeštenog u akumulatorskom registru A. Sledeće stanje je stanje `CHECK_A` gde se proverava znak dobijene vrednosti u akumulatorskom registru. U slučaju da je ona veća od nule prelazi se u stanje `SET_X0` gde se postavlja bit X_0 na vrednost 1. U suprotnom slučaju se prelazi u stanje `ADD` gde se bit X_0 postavlja na vrednost 0 i obnavlja se vrednost u akumulatorskom registru sabiranjem delitelja i akumulatora. Iz stanja `SET_X0` i `ADD` se prelazi u stanje `CNT`, gde se proverava da li je brojač stigao do kraja ciklusa brojanja, kada se automat vraća u inicijalno stanje `IDLE`. Ako brojač nije stigao do kraja ciklusa brojanja prelazi se u stanje `SHIFT` i postupak se ponavlja do kraja ciklusa brojanja. Tada se završava izvršenje algoritma delenja neoznačenih celih brojeva.

Upravljački automat u ovom slučaju ima sledeće ulazne signale:

- `iCLK` takt signal,
- `inRESET` signal za postavljanje automata u inicijalno stanje, aktivan na niskom nivou i sinhron sa signalom takta,
- `iSTART` impuls za početak izvršenja algoritma množenja,
- `iA_LT_0` visoka logička vrednost na ovom signalu ukazuje da je vrednost u akumulatorskom registru manja od nule. U suprotnom slučaju je vrednost u akumulatorskom registru veća od nule.
- `iCNT_0` indikacija kraja ciklusa brojanja brojača ciklusa množenja, aktivna na visokom nivou.

Potrebno je generisati sledeće upravljačke signale:

- `oLOAD_CNT` impuls za upis početne vrednosti brojača,
- `oLOAD_X` impuls za upis deljenika,
- `oLOAD_Y` impuls za upis delitelja,
- `oLOAD_A` impuls za upis rezultata sabiranja u akumulator,
- `onRST_A` impuls za brisanje sadržaja akumulatora, aktivan na niskom nivou,
- `oCE` dozvola pomeranja i dozvola brojanja,
- `oADD_SUB` upravljački signal za određivanje koja će se operacija izvršiti. Ako je ovaj signal na niskom nivou izvršiće se sabiranje, dok će se oduzimanje izvršiti u suprotnom slučaju,

– oX_SEL — vektor za odabiranje vrednosti koja će se upisati u registar X
= 00 → upisuje se vredost deljenika
= 01 → bit X_0 se postavlja na nulu
= 10 → bit X_0 se postavlja na jedinicu

– oREADY — izlazni signal koji govori da je u toku izvršenje algoritma, ako je njegova vrednost na niskom logičkom nivou. U suprotnom, oREADY=1, sistem je spreman za izvršenje algoritma.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY CONTROL IS PORT (
  iCLK, inRESET: IN  STD_LOGIC;
  iSTART, iA_LT_0, iCNT_0: IN  STD_LOGIC;
  oLOAD_X, oLOAD_CNT, oLOAD_A,
  onRST_A, oLOAD_Y, oCE,
  oADD_SUB, oREADY: OUT STD_LOGIC;
  oX_SEL:           OUT STD_LOGIC_VECTOR(1 DOWNTO 0) );
END CONTROL;

ARCHITECTURE ARH_CONTROL OF CONTROL IS
  -- tip podataka za identifikaciju stanja automata
  TYPE tSTANJA IS
               (IDLE, INIT, SHIFT, SUB, CHECK_A, ADD, SET_X0, CNT);
  -- deklaracija signala koji sadrze informaciju o
  -- tekucem i sledecem (narednom) stanju
  SIGNAL sSTANJE, sSLEDECE_STANJE : tSTANJA;
BEGIN

  -- kombinaciona mreza koja na osnovu trenutnog stanja
  -- i vrednosti ulaza generise kod narednog stanja
  PROCESS (iSTART, iA_LT_0, iCNT_0, sSTANJE) BEGIN
    CASE sSTANJE IS
      WHEN IDLE =>
        IF (iSTART = '0') THEN
          sSLEDECE_STANJE <= IDLE;
        ELSE
          sSLEDECE_STANJE <= INIT;
        END IF;
      WHEN INIT =>
        sSLEDECE_STANJE <= SHIFT;
      WHEN SHIFT =>
        sSLEDECE_STANJE <= SUB;
      WHEN SUB =>
        sSLEDECE_STANJE <= CHECK_A;
      WHEN CHECK_A =>
        IF (iA_LT_0 = '0') THEN
          -- akumulator nije manji od nule
          sSLEDECE_STANJE <= SET_X0;
        ELSE
          -- akumulator je manji od nule
          sSLEDECE_STANJE <= ADD;
```

```
        END IF;
      WHEN SET_X0 =>
        sSLEDECE_STANJE <= CNT;
      WHEN ADD =>
        sSLEDECE_STANJE <= CNT;
      WHEN CNT =>
        IF (iCNT_0 = '0') THEN
          -- vrednost brojaca je veca od nule
          sSLEDECE_STANJE <= SHIFT;
        ELSE
          -- brojac je stigao do nule
          sSLEDECE_STANJE <= IDLE;
        END IF;
    END CASE;
  END PROCESS;

  -- kombinaciona mreza za odredjivanje vrednosti
  -- izlaznog vektora na osnovu trenutnog stanja
  PROCESS (sSTANJE) BEGIN
    CASE sSTANJE IS
      WHEN IDLE =>
        oLOAD_X   <= '0'; oLOAD_Y <= '0';
        oLOAD_A   <= '0'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '0';
        oADD_SUB  <= '0'; oX_SEL  <= "00";
        oREADY    <= '1';
      WHEN INIT =>
        oLOAD_X   <= '1'; oLOAD_Y <= '1';
        oLOAD_A   <= '0'; onRST_A <= '0';
        oLOAD_CNT <= '1'; oCE     <= '0';
        oADD_SUB  <= '0'; oX_SEL  <= "00";
        oREADY    <= '0';
      WHEN SHIFT =>
        oLOAD_X   <= '0'; oLOAD_Y <= '0';
        oLOAD_A   <= '0'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '1';
        oADD_SUB  <= '0'; oX_SEL  <= "00";
        oREADY    <= '0';
      WHEN SUB =>
        oLOAD_X   <= '0'; oLOAD_Y <= '0';
        oLOAD_A   <= '1'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '0';
        oADD_SUB  <= '1'; oX_SEL  <= "00";
        oREADY    <= '0';
      WHEN CHECK_A =>
        oLOAD_X   <= '0'; oLOAD_Y <= '0';
        oLOAD_A   <= '0'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '0';
        oADD_SUB  <= '0'; oX_SEL  <= "00";
        oREADY    <= '0';
      WHEN ADD =>
        oLOAD_X   <= '1'; oLOAD_Y <= '0';
        oLOAD_A   <= '1'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '0';
        oADD_SUB  <= '0'; oX_SEL  <= "01";
        oREADY    <= '0';
      WHEN SET_X0 =>
```

```
        oLOAD_X   <= '1'; oLOAD_Y <= '0';
        oLOAD_A   <= '0'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '0';
        oADD_SUB  <= '0'; oX_SEL  <= "10";
      WHEN CNT =>
        oLOAD_X   <= '0'; oLOAD_Y <= '0';
        oLOAD_A   <= '0'; onRST_A <= '1';
        oLOAD_CNT <= '0'; oCE     <= '0';
        oADD_SUB  <= '0'; oX_SEL  <= "00";
        oREADY    <= '0';
    END CASE;
  END PROCESS;

  -- flip-flopovi za smestanje koda
  -- trenutnog stanja; postavljanje pocetnog
  -- stanja je sinhrono sa signalom takta
  PROCESS (iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK='1') THEN
      IF (inRESET = '0') THEN -- sinhroni reset
        sSTANJE <= IDLE;
      ELSE
        sSTANJE <= sSLEDECE_STANJE;
      END IF;
    END IF;
  END PROCESS;
END ARH_CONTROL;
```

MODUL DELITELJA:

U ovom modulu se instanciraju svi prethodno opisani moduli i međusobno povezuju u skladu sa blok šemom digitalnog sistema za delenje celih neoznačenih brojeva, Slika 8.39.

Prvo je potrebno formirati pakovanje sa opisom sprega svih modula koji se instanciraju:

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

PACKAGE DELITELJ_PKG IS
  COMPONENT REG
    GENERIC (
      pWIDTH: integer := 4
    );
    PORT (
      iCLK, inCLR: IN  STD_LOGIC;
      iCE: IN  STD_LOGIC;
      iD:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
      oQ:  OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
    );
  END COMPONENT;

  COMPONENT SHIFT_REG
    GENERIC (
      pWIDTH: integer := 4
    );
    PORT (
```

```
      iCLK, inCLR: IN  STD_LOGIC;
      iLOAD, iSE, iDSL: IN  STD_LOGIC;
      iD: IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
      oQ: OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
    );
  END COMPONENT;

  COMPONENT BROJAC
    GENERIC (
      pWIDTH: integer := 4
    );
    PORT (
      iCLK    : IN  STD_LOGIC;
      inRESET,
      iEN, iP : IN  STD_LOGIC;
      iDATA   : IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
      oQ      : OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
      oTSE    : OUT STD_LOGIC
    );
  END COMPONENT;

  COMPONENT ADDER
    GENERIC (
      pWIDTH: integer := 4
    );
    PORT (
      iX, iY:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
      iSEL:    IN  STD_LOGIC;
      oZ:      OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
    );
  END COMPONENT;

  COMPONENT CONTROL
    PORT (
      iCLK, inRESET: IN  STD_LOGIC;
      iSTART, iA_LT_0, iCNT_0: IN  STD_LOGIC;
      oLOAD_Y, oLOAD_CNT, oLOAD_A,
      onRST_A, oLOAD_X, oCE,
      oADD_SUB, oREADY: OUT STD_LOGIC;
      oX_SEL:           OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
    );
  END COMPONENT;

END DELITELJ_PKG;
```

Nakon formiranja pakovanja sa opisom sprega svih komponenti koje se instanciraju prelazi se na opis entiteta projektovanog digitalnog sistema. Projektovani modul delitelja sadrži sledeće ulazno/izlazne signale:

- `iCLK` takt signal,
- `inCLR` signal za brisanje sadržaja svih registara i brojača, aktivan na niskom nivou i sinhron sa signalom takta,
- `iSTART` impuls za početak izvršenja algoritma delenja,
- `iX` deljenik, četvorobitni ulazni vektor,

- iY delitelj, četvorobitni ulazni vektor,
- oZ količnik, četvorobitni izlazni vektor,
- oO ostatak, četvorobitni izlazni vektor i
- oREADY izlazni signal koji govori da je u toku izvršenje algoritma, ako je njegova vrednost na niskom logičkom nivou. U suprotnom, oREADY=1, sistem je spreman za izvršenje algoritma.

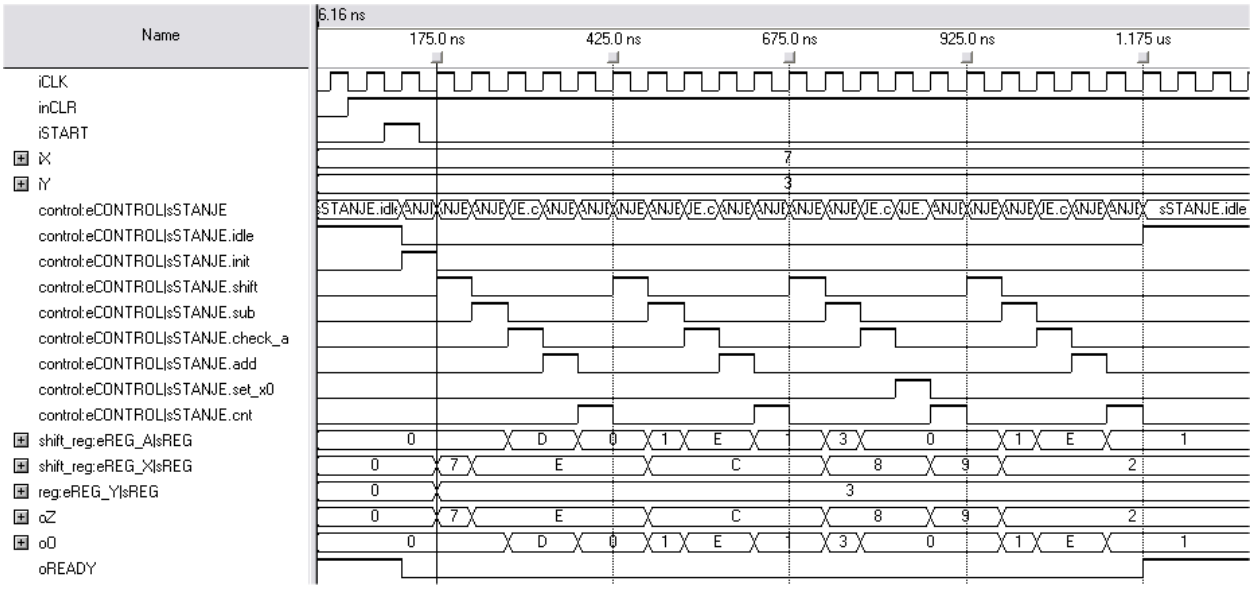
```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;
  -- ukljucivanje pakovanja sa komponentama
  USE work.DELITELJ_PKG.all;

ENTITY DELITELJ IS
  PORT (
    iCLK, inCLR: IN  STD_LOGIC;
    iSTART: IN  STD_LOGIC;
    iX, iY: IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    oZ, oO: OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    oREADY: OUT STD_LOGIC );
END DELITELJ;

ARCHITECTURE ARH_DELITELJ OF DELITELJ IS
  -- konstanta nula
  CONSTANT cZERO: STD_LOGIC := '0';
  -- broj bita sa kojima se reprezentuju operandi
  CONSTANT cNUM_BITS: STD_LOGIC_VECTOR(2 DOWNTO 0) := "100";

  -- registar deljenika i delitelja
  SIGNAL sY, sX: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- ulazni vektor u registar za smestanje delitelja
  SIGNAL sDATA:  STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- akumulator i rezultat sabiraca
  SIGNAL sA, sR: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- brojac bita
  SIGNAL sCNT: STD_LOGIC_VECTOR(2 DOWNTO 0);
  -- upravljacki signali za registre i brojac
  SIGNAL sLOAD_Y, sLOAD_X, sLOAD_A,
         sLOAD_CNT, snRST_A, sCE,
         sCNT_TC, snCLR_A, sADD_SUB: STD_LOGIC;
  SIGNAL sX_SEL: STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
  -- registar za smestanje delitelja
  eREG_Y: REG
    GENERIC MAP (pWIDTH=>4)
    PORT    MAP (iCLK=>iCLK, inCLR=>inCLR, iCE=>sLOAD_Y,
                 iD=>iY, oQ=>sY);

  -- multiplekser za odabir vrednosti
  -- koja se upisuje u registar deljenika
  PROCESS (iX, sX_SEL) BEGIN
    CASE sX_SEL IS
                  -- propusti deljenik
      WHEN "00"   => sDATA <= iX;
                  -- resetuj bit 0
```

```
        WHEN "01"   => sDATA <= (sX(3 DOWNTO 1) & '0');
                       -- postavi na 1 bit 0
        WHEN OTHERS => sDATA <= (sX(3 DOWNTO 1) & '1');
      END CASE;
    END PROCESS;

    -- registar za smestanje deljenika
    eREG_X: SHIFT_REG
      GENERIC MAP (pWIDTH=>4)
      PORT    MAP (iCLK=>iCLK, inCLR=>inCLR,
                   iLOAD=>sLOAD_X, iSE=>sCE, iDSL=>cZERO,
                   iD=>sDATA, oQ=>sX);

    -- akumulator za smestanje rezultata i izlaznog prenosa
    snCLR_A <= inCLR AND snRST_A; -- resetovanje akumulatora
    eREG_A: SHIFT_REG
      GENERIC MAP (pWIDTH=>4)
      PORT    MAP (iCLK=>iCLK, inCLR=>snCLR_A,
                   iLOAD=>sLOAD_A, iSE=>sCE, iDSL=>sX(3),
                   iD=>sR, oQ=>sA);

    -- sabirac/oduzimac akumulisane vrednosti
    -- u akumulatoru i delitelja
    -- rezultat se smesta nazad u akumulator
    eADDER: ADDER
      GENERIC MAP (pWIDTH=>4)
      PORT MAP (iX=>sA, iY=>sY, iSEL=>sADD_SUB, oZ=>sR);

    -- brojac ciklusa (broja obradjenih bita) tokom delenja
    eCNT: BROJAC
      GENERIC MAP (pWIDTH=>3)
      PORT    MAP (iCLK=>iCLK, inRESET=>inCLR, iEN=>sCE,
                   iP=>sLOAD_CNT, iDATA=>cNUM_BITS,
                   oQ=>sCNT, oTSE=>sCNT_TC);

    -- automat ze generisanje upravljackih signala
    eCONTROL: CONTROL
      PORT MAP (iCLK=>iCLK, inRESET=>inCLR,
                iSTART=>iSTART, iA_LT_0=>sA(3), iCNT_0=>sCNT_TC,
                oLOAD_Y=>sLOAD_Y, oLOAD_X=>sLOAD_X, oLOAD_A=>sLOAD_A,
                onRST_A=>snRST_A, oLOAD_CNT=>sLOAD_CNT, oCE=>sCE,
                oX_SEL=>sX_SEL, oADD_SUB=>sADD_SUB, oREADY=>oREADY);

    oZ <= sX; -- rezultat
    oO <= sA; -- ostatak
  END ARH_DELITELJ;
```

Za registar X pored operacije pomeranja u levo, treba da se omogući upis vrednosti deljenika, postavljanje bita X_0 na vrednost nula ili na vrednost 1. Pomeranje sadržaja registra u levo je realizovano instanciranjem odgovarajućeg pomeračkog registra i pravovremenim generisanjem signala dozvole pomeranja. Upis delitelja i postavljanje odgovarajuće vrednosti bita najniže važnosti je realizovano uvođenjem multipleksera čiji se izlaz, signal `sDATA`, upisuje u registar preko ulaznog vektora `iD`:

```
PROCESS (iX, sX_SEL) BEGIN
  CASE sX_SEL IS
                -- propusti deljenik
    WHEN "00"   => sDATA <= iX;
                -- resetuj bit 0
    WHEN "01"   => sDATA <= (sX(3 DOWNTO 1) & '0');
                -- postavi na 1 bit 0
    WHEN OTHERS => sDATA <= (sX(3 DOWNTO 1) & '1');
  END CASE;
END PROCESS;
```

U slučaju da upravljački signal `sX_SEL` ima vrednost "00" generisanjem signala dozvole upisa `sLOAD_X` u registar se upisuje vrednost deljenika prisutna na ulaznom vektoru `iX`. Kada signal `sX_SEL` ima vrednost "01" u registar se na mesto sa indeksom nula upisuje vrednost 0, dok se vrednost preostala tri bita zadržava upisom njihove vrednosti. Istom metodom se za vrednost "10" signala `sX_SEL` postavlja vrednost 1 na bit najmanje važnosti i zadržava se stanje preostalih bita registra X.



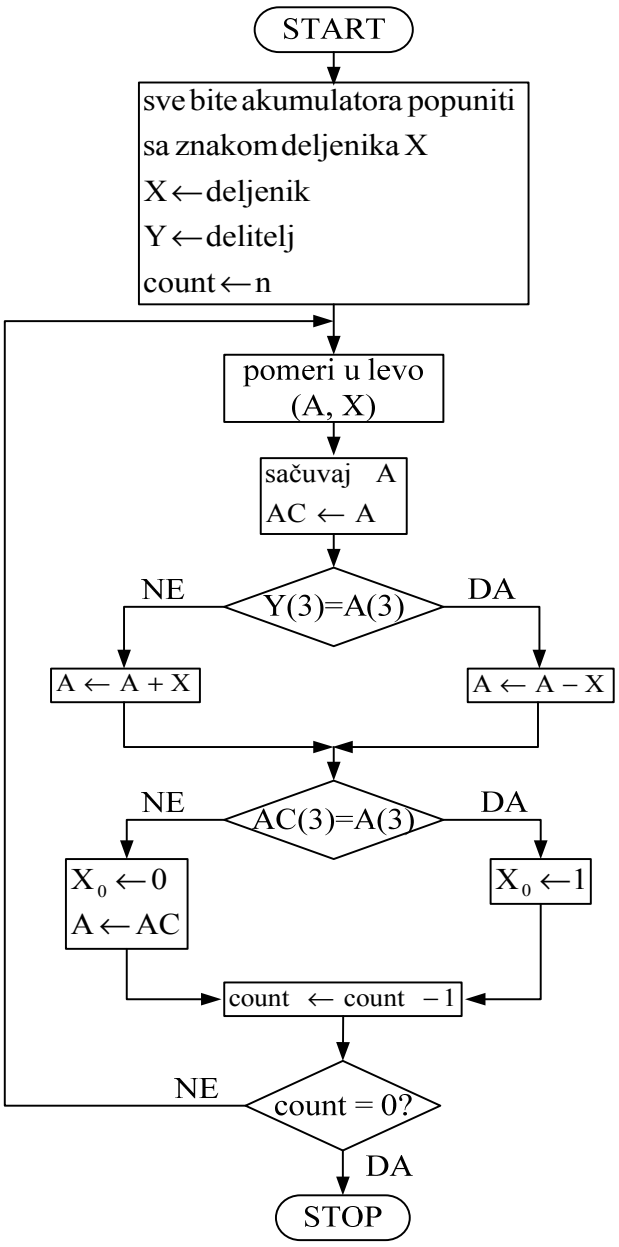
Slika 8.41: Simulacija rada realizovanog delitelja

Slika 8.41 prikazuje vremenski dijagram izvršenja operacije delenja 7 sa 3 prema zadatom algoritmu. Nakon postavljanja inicijalnih vrednosti registara u stanju sa kodom 01 (`INIT`) u vremenskom trenutku 175μs započinje prvi ciklus delenja koji se završava u vremenskom trenutku 425μs kada započinje drugi ciklus. Drugi ciklus traje do vremenskog trenutka 675μs. U ova prva dva ciklusa algoritam prolazi kroz stanja sa kodovima 02 (`SHIFT`), 03 (`SUB`), 04 (`CHECK_A`), 05 (`ADD`) i 07 (`CHECK_CNT`). U trećem ciklusu, koji se završava u vremenskom trenutku 925μs, algoritam umesto kroz stanje `ADD` prlazi kroz stanje `SET_Q0`. Četvrti ciklus prolazi kroz ista stanja kao i prva dva ciklusa i završava se u vremenskom trenutku 1,175μs kada se i završava izvršenje algoritma (`oREADY=1`).

Rezultat izvršenja operacije delenja 7 sa 3 se vidi na izlaznim vektorima `oZ=2` i `oO=1` koji prikazuju količnik i ostatak respektivno.

8.11 ZADATAK:

Izvršiti sintezu digitalnog sistema za delenje celih označenih brojeva prema algoritmu koji prikazuje Slika 8.42. Digitalni sistem treba projektovati za potrebe delenja četvorobitnih brojeva.



Slika 8.42: Blok dijagram algoritma za delenje celih označenih brojeva

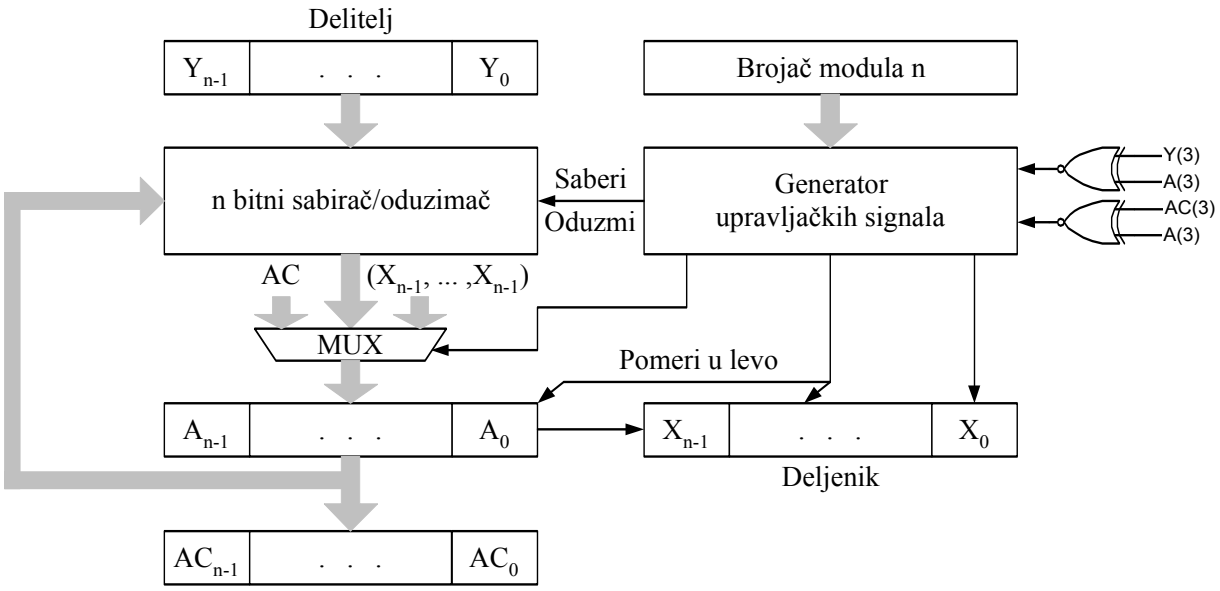
U rešenju treba prikazati blok šemu digitalnog sistema, graf prelazaka stanja generatora upravljačkih signala, kao i VHDL kod kompletnog digitalnog sistema.

REŠENJE:

Prikazani algoritam predstavlja modifikaciju algoritma iz prethodnog zadatka za delenje pozitivnih i negativnih brojeva predstavljenih u drugom komplementu. Modifikacija se sastoji u sledećem:

1. Deljenik upisati u registar Y, a delitelj u registre A i X tako da delitelj predstavlja broj u drugom komplementu. Tako na primer četvorobitna vrednost 0101 postaje 00000101, i 1010 postaje 11111010. To dalje znači da u registar X treba upisati četvorobitnu vrednost delitelja, a sve bite akumulatorskog registra popuniti sa znakom delitelja.
2. Izvršiti pomeranje registara A i X za jedno mesto u levo.
3. Ako vrednost u registru A i deljenik Y imaju iste znakove treba izvršiti oduzimanje deljenika od akumulatora (A←A- Y). U suprotnom izvršava se sabiranje akumulatora i deljenika (A←A+ Y).
4. Prethodna operacija se smatra za uspešnu ako je vrednost u akumulatorskom registru zadržala isti znak kao i pre izvršenja operacije. U tom slučaju se bit najniže važnosti registra X postavlja na vrednost 1 (X_0←1). Inače se bit X_0 postavlja na vrednost 0 (X_0←0) i izvršava se obnavljanje sadržaja akumulatorskog registra.
5. Koraci 2 do 4 se ponavljaju za sve bite deljenika X.
6. Na kraju izvršenja algoritma ostatak je smešten u registru A dok je pozitivna vrednost količnika smeštena u registru X. U slučaju da deljenik i delitelj imaju različite znakove količnik je drugi komplement od vrednosti smeštene u registru X.

Za realizaciju traženog algoritma mogu se primeniti isti principi kao i za realizaciju delitelja iz prethodnog zadatka. Međutim, moraju se uvesti i određene izmene koje se vide u opisu algoritma deljenja celih označenih brojeva. Prva izmena se odnosi na potrebu za pamćenjem sadržaja akumulatorskog registra pre izvršenja operacije sabiranja/oduzimanja. Zbog toga se uvodi dodatni registar koji je označen simbolom AC. Akumulatorski registar može da primi vrednost iz ukupno tri različita izvora. To su rezultujući vektor sabirača/oduzimača, stara vrednost akumulatorskog registra smeštena u registru AC i popunjavanje svih bita akumulatorskog registra sa znakom deljenika X prilikom inicijalizacije digitalnog sistema. Zbog toga je potrebno uvesti multiplekser koji će u datom trenutku generisati odgovarajuću vrednost. Potrebno je i uvođenje kombinacionih mreža za poređenje znakova vrednosti u akumulatorskom registru A sa znakom delitelja Y i sa znakom stare vrednosti akumulatorskog registra smeštene u registru AC. Poslednji dodatak je vezan za proveru znaka rezultata. U slučaju da su deljenik i delitelj različitih znakova rezultat je negativan, pa je potrebno generisati drugi komplement od dobijenog količnika. U suprotnom slučaju, na izlazni vektor rezultat se prosleđuje dobijeni količnik koji je smešten u registru X. Ostali elementi digitalnog sistema za delenje celih označenih brojeva je su isti kao u prethodnom zadatku, kao što prikazuje Slika 8.43.



Slika 8.43: Blok šema digitalnog sistema za delenje celih označenih brojeva

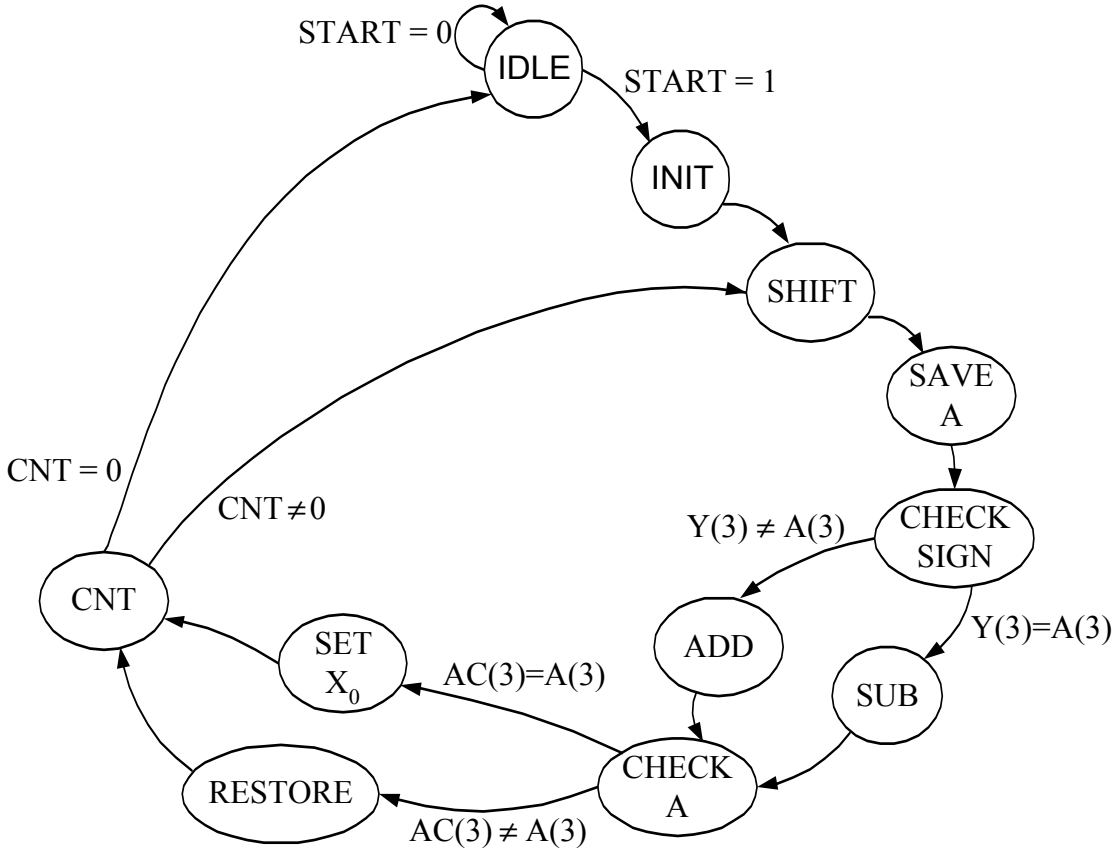
Procedura delenja prema ovom algoritmu se može ilustrovati na primeru delenja brojeva 7 i 3. Deljenik ima vrednost $7_{DEC}=0411_{BIN}$, dok je delitelj $3_{DEC}=0011_{BIN}$. Njihov količnik je $2_{DEC}=0010_{BIN}$, a ostatak pri delenju je $1_{DEC}=0001_{BIN}$. Prema tome, na kraju izvršenja algoritma u registru A treba da je vrednost 0001_{BIN}, a u registru X vrednost 0010_{BIN}. Za date vrednosti deljenika i delitelja algoritam funcioniše na sledeći način:

A	X	Y		
0000	0111	0011	Inicijalne vrednosti	
0000	1110	0011	Pomeranje	Prvi ciklus
1101	1110	0011	Oduzimanje	
0000	1110	0011	Obnavljanje	
0001	1100	0011	Pomeranje	Drugi ciklus
1110	1100	0011	Oduzimanje	
0001	1100	0011	Obnavljanje	
0011	1000	0011	Pomeranje	Treći ciklus
0000	1000		Oduzimanje	
0000	1001	0011	$X_0 \leftarrow 1$	
0001	0010	0011	Pomeranje	Četvti ciklus
1110	0010	0011	Oduzimanje	
0001	0010	0011	Obnavljanje	

Za VHDL implementaciju opisanog digitalnog sistema mogu se iskoristiti sve komponente delitelja iz prethodnog zadatka, izuzev automata za generisanje upravljačkih signala.

MODUL ZA GENERISANJE UPRAVLJAČKIH SIGNALA:

Ovaj modul treba da na osnovu zadatog blok dijagram algoritma delenja označenih celih brojeva, Slika 8.42, generiše odgovarajuće upravljačke signale za registre, brojač i sabirač. Modul za generisanje upravljačkih signala se realizuje pomoću Murovog automata. Radi lakše implementacije formira se graf prelazaka stanja, Slika 8.44, gde se u svakom stanju generišu odgovarajući upravljački signali u skladu sa potrebnom operacijom u tom vremenskom trenutku.



Slika 8.44: Graf prelazaka stanja automata za generisanje upravljačkih signala

Automat se inicijalno nalazi u stanju `IDLE`. U slučaju da se na ulazu pojavi impuls za početak izvršenja algoritma, `START=1`, automat prelazi u stanje `INIT` gde se izvršava inicijalizacija stanja svih registara i brojača. Nakon toga prelazi se u stanje `SHIFT` gde se izvršava pomeranje sadržaja registara A i Y. Takođe, u ovom stanju se i dekrementuje sadržaj brojača. To je mala modifikacija datog algoritma, ali ona ne utiče na njegovo izvršenje pošto se vrednost brojača proverava tek na kraju ciklusa. Modifikacija je urađena radi zadržavanja jedinstvenog upravljačkog signala za menjanje sadržaja pomeračkih registara i brojača, kao i kod realizacije množača iz prethodna dva zadatka. Bez ove modifikacije bilo bi potrebno uvesti dva nezavisna upravljačka signala za dozvolu pomeranja pomeračkih registara i za dozvolu brojanja brojača.

Iz stanja `SHIFT` se prelazi u stanje `SAVE_A` gde će se sačuvati vrednost akumulatorskog registra pre izvršenja operacije sabiranja ili oduzimanja. Sledeće

stanje, CHECK_SIGN, služi za poređenje znakova akumulatorskog registra A i delitelja Y. U slučaju da su znakovi isti prelazi se u stanje SUB gde se generišu upravljački signali za oduzimanje delitelja Y od sadržaja u akumulatorskom registru A. U suprotnom slučaju se prelazi u stanje ADD gde se sabiraju vrednosti akumulatorskog registra A i delitelja Y. Iz stanja SUB i ADD se bezuslovno prelazi u stanje CHECK_A gde se poredi znak dobijene vrednosti u akumulatorskom registru sa znakom njegove prethodno sačuvane vrednosti u registru AC. U slučaju da su znakovi isti prelazi se u stanje SET_X0 gde se postavlja bit X_0 na vrednost 1. U suprotnom slučaju se prelazi u stanje RESTORE gde se bit X_0 postavlja na vrednost 0 i obnavlja se vrednost u akumulatorskom registru upisivanjem sačuvanog sadržaja u registru AC. Iz stanja SET_X0 i RESTORE se prelazi u stanje CNT, gde se proverava da li je brojač stigao do kraja ciklusa brojanja, kada se automat vraća u inicijalno stanje IDLE. Ako brojač nije stigao do kraja ciklusa brojanja, prelazi se u stanje SHIFT i postupak se ponavlja do kraja ciklusa brojanja. Tada se završava izvršenje algoritma delenja neoznačenih celih brojeva.

Upravljački automat u ovom slučaju ima sledeće ulazne signale:

- iCLK takt signal,
- inRESET signal za postavljanje automata u inicijalno stanje, aktivan na niskom nivou i sinhron sa signalom takta,
- iSTART impuls za početak izvršenja algoritma množenja,
- iAY_EQ visoka logička vrednost na ovom signalu ukazuje su da vrednost u akumulatorskom registru A i delitelj Y istog znaka. U suprotnom slučaju znakovi su različiti.
- iAAC_EQ visoka logička vrednost na ovom signalu ukazuje da su vrednost u akumulatorskom registru A i vrednost u registru AC istog znaka. U suprotnom slučaju znakovi su različiti.
- iCNT_0 indikacija kraja ciklusa brojanja brojača ciklusa množenja, aktivna na visokom nivou.

Potrebno je generisati sledeće upravljačke signale:

- oLOAD_CNT impuls za upis početne vrednosti brojača,
- oLOAD_X impuls za upis množioca,
- oLOAD_Y impuls za upis množitelja,
- oLOAD_A impuls za upis rezultata sabiranja u akumulator,
- oLOAD_AC impuls za upis sadržaja akumulatora u registar AC,
- oCE dozvola pomeranja i dozvola brojanja,
- oADD_SUB upravljački signal za određivanje koja će se operacija izvršiti. Ako je ovaj signal na niskom nivou izvršiće se sabiranje, dok će se oduzimanje izvršiti u suprotnom slučaju,

- oX_SEL — vektor za odabiranje vrednosti koja će se upisati u registar X
  = 00 → upisuje se vredost delitelja
  = 01 → bit X_0 se postavlja na nulu
  = 10 → bit X_0 se postavlja na jedinicu
- oA_SEL — vektor za odabiranje vrednosti koja će se upisati u registar A
  = 00 → upisuje se znak deljenika na sve pozicije u registru,
  inicijalizacija sadržaja registra
  = 01 → upisuje se rezultat sabirača/oduzimača,
  izvršenje aritmetičke operacije
  = 10 → upisuje se sadržaj registra AC,
  obnavljanje sadržaja akumulatora
- oREADY — izlazni signal koji govori da je u toku izvršenje algoritma, ako je njegova vrednost na niskom logičkom nivou. U suprotnom, oREADY=1, sistem je spreman za izvršenje algoritma.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY CONTROL IS PORT (
  iCLK, inRESET: IN  STD_LOGIC;
  iSTART, iCNT_0, iAY_EQ, iAAC_EQ: IN  STD_LOGIC;
  oLOAD_Y, oLOAD_X, oLOAD_CNT,
  oLOAD_A, oLOAD_AC,
  oCE, oADD_SUB, oREADY: OUT STD_LOGIC;
  oX_SEL, oA_SEL: OUT STD_LOGIC_VECTOR(1 DOWNTO 0) );
END CONTROL;

ARCHITECTURE ARH_CONTROL OF CONTROL IS
  -- tip podataka za identifikaciju stanja automata
  TYPE tSTANJA IS (IDLE, INIT, SHIFT, SAVE_A, CHECK_SIGN,
                   SUB, ADD, CHECK_A, RESTORE, SET_X0, CNT);
  -- deklaracija signala koji sadrze informaciju o
  -- tekucem i sledecem (narednom) stanju
  SIGNAL sSTANJE, sSLEDECE_STANJE : tSTANJA;
BEGIN

  -- kombinaciona mreza koja na osnovu trenutnog stanja
  -- i vrednosti ulaza generise kod narednog stanja
  PROCESS (iSTART, iAY_EQ, iAAC_EQ, iCNT_0, sSTANJE) BEGIN
    CASE sSTANJE IS
      WHEN IDLE =>
        IF (iSTART = '0') THEN
          sSLEDECE_STANJE <= IDLE;
        ELSE
          sSLEDECE_STANJE <= INIT;
        END IF;
      WHEN INIT =>
```

```
        sSLEDECE_STANJE <= SHIFT;
      WHEN SHIFT =>
        sSLEDECE_STANJE <= SAVE_A;
      WHEN SAVE_A =>
        sSLEDECE_STANJE <= CHECK_SIGN;
      WHEN CHECK_SIGN =>
        IF (iAY_EQ = '0') THEN
          -- akumulator i delitelj imaju razlicite znakove
          sSLEDECE_STANJE <= ADD;
        ELSE
          -- akumulator i delitelj imaju iste znakove
          sSLEDECE_STANJE <= SUB;
        END IF;
      WHEN SUB =>
        sSLEDECE_STANJE <= CHECK_A;
      WHEN ADD =>
        sSLEDECE_STANJE <= CHECK_A;
      WHEN CHECK_A =>
        IF (iAAC_EQ = '0') THEN
          -- akumulator i njegova vrednost pre pomeranja
          -- imaju razlicite znakove
          sSLEDECE_STANJE <= RESTORE;
        ELSE
          -- akumulator i njegova vrednost pre pomeranja
          -- imaju iste znakove
          sSLEDECE_STANJE <= SET_X0;
        END IF;
      WHEN RESTORE =>
        sSLEDECE_STANJE <= CNT;
      WHEN SET_X0 =>
        sSLEDECE_STANJE <= CNT;
      WHEN CNT =>
        IF (iCNT_0 = '0') THEN
          -- vrednost brojaca je veca od nule
          sSLEDECE_STANJE <= SHIFT;
        ELSE
          -- brojac je stigao do nule
          sSLEDECE_STANJE <= IDLE;
        END IF;
    END CASE;
  END PROCESS;

  -- kombinaciona mreza za odredjivanje vrednosti
  -- izlaznog vektora na osnovu trenutnog stanja
  PROCESS (sSTANJE) BEGIN
    CASE sSTANJE IS
      WHEN IDLE =>
        oLOAD_X   <= '0';  oLOAD_Y  <= '0';
        oLOAD_A   <= '0';  oLOAD_AC <= '0';
        oLOAD_CNT <= '0';  oCE      <= '0';
        oA_SEL    <= "00"; oX_SEL   <= "00";
        oADD_SUB  <= '0';  oREADY   <= '1';
      WHEN INIT =>
        oLOAD_X   <= '1';  oLOAD_Y  <= '1';
        oLOAD_A   <= '1';  oLOAD_AC <= '0';
        oLOAD_CNT <= '1';  oCE      <= '0';
        oA_SEL    <= "00"; oX_SEL   <= "00";
        oADD_SUB  <= '0';  oREADY   <= '0';
```

```
      WHEN SHIFT =>
        oLOAD_X   <= '0';  oLOAD_Y  <= '0';
        oLOAD_A   <= '0';  oLOAD_AC <= '0';
        oLOAD_CNT <= '0';  oCE      <= '1';
        oA_SEL    <= "00"; oX_SEL   <= "00";
        oADD_SUB  <= '0';  oREADY   <= '0';
      WHEN SAVE_A =>
        oLOAD_X   <= '0';  oLOAD_Y  <= '0';
        oLOAD_A   <= '0';  oLOAD_AC <= '1';
        oLOAD_CNT <= '0';  oCE      <= '0';
        oA_SEL    <= "00"; oX_SEL   <= "00";
        oADD_SUB  <= '0';  oREADY   <= '0';
      WHEN CHECK_SIGN =>
        oLOAD_X   <= '0';  oLOAD_Y  <= '0';
        oLOAD_A   <= '0';  oLOAD_AC <= '0';
        oLOAD_CNT <= '0';  oCE      <= '0';
        oA_SEL    <= "00"; oX_SEL   <= "00";
        oADD_SUB  <= '0';  oREADY   <= '0';
      WHEN SUB=>
        oLOAD_X   <= '0';  oLOAD_Y  <= '0';
        oLOAD_A   <= '1';  oLOAD_AC <= '0';
        oLOAD_CNT <= '0';  oCE      <= '0';
        oA_SEL    <= "01"; oX_SEL   <= "00";
        oADD_SUB  <= '1';  oREADY   <= '0';
      WHEN ADD =>
        oLOAD_X   <= '0';  oLOAD_Y  <= '0';
        oLOAD_A   <= '1';  oLOAD_AC <= '0';
        oLOAD_CNT <= '0';  oCE      <= '0';
        oA_SEL    <= "01"; oX_SEL   <= "00";
        oADD_SUB  <= '0';  oREADY   <= '0';
      WHEN CHECK_A =>
        oLOAD_X   <= '0';  oLOAD_Y  <= '0';
        oLOAD_A   <= '0';  oLOAD_AC <= '0';
        oLOAD_CNT <= '0';  oCE      <= '0';
        oA_SEL    <= "00"; oX_SEL   <= "00";
        oADD_SUB  <= '0';  oREADY   <= '0';
      WHEN RESTORE =>
        oLOAD_X   <= '1';  oLOAD_Y  <= '0';
        oLOAD_A   <= '1';  oLOAD_AC <= '0';
        oLOAD_CNT <= '0';  oCE      <= '0';
        oA_SEL    <= "10"; oX_SEL   <= "01";
        oADD_SUB  <= '0';  oREADY   <= '0';
      WHEN SET_X0 =>
        oLOAD_X   <= '1';  oLOAD_Y  <= '0';
        oLOAD_A   <= '0';  oLOAD_AC <= '0';
        oLOAD_CNT <= '0';  oCE      <= '0';
        oA_SEL    <= "00"; oX_SEL   <= "10";
        oADD_SUB  <= '0';  oREADY   <= '0';
      WHEN CNT =>
        oLOAD_X   <= '0';  oLOAD_Y  <= '0';
        oLOAD_A   <= '0';  oLOAD_AC <= '0';
        oLOAD_CNT <= '0';  oCE      <= '0';
        oA_SEL    <= "00"; oX_SEL   <= "00";
        oADD_SUB  <= '0';  oREADY   <= '0';
    END CASE;
  END PROCESS;
```

```
    -- flip-flopovi za smestanje koda
    -- trenutnog stanja; postavljanje pocetnog
    -- stanja je sinhrono sa signalom takta
    PROCESS (iCLK) BEGIN
      IF (iCLK'EVENT AND iCLK='1') THEN
        IF (inRESET = '0') THEN -- sinhroni reset
          sSTANJE <= IDLE;
        ELSE
          sSTANJE <= sSLEDECE_STANJE;
        END IF;
      END IF;
    END PROCESS;
  END ARH_CONTROL;
```

MODUL DELITELJA:

U ovom modulu se instanciraju svi prethodno opisani moduli i međusobno povezuju u skladu sa blok šemom digitalnog sistema za delenje celih označenih brojeva, Slika 8.43.

Prvo je potrebno formirati pakovanje sa opisom sprega svih modula koji se instanciraju:

```
  LIBRARY ieee;
    USE ieee.std_logic_1164.all;

  PACKAGE DELITELJ_PKG IS
    COMPONENT REG
      GENERIC (
        pWIDTH: integer := 4
      );
      PORT (
        iCLK, inCLR: IN  STD_LOGIC;
        iCE: IN  STD_LOGIC;
        iD:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
        oQ:  OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
      );
    END COMPONENT;

    COMPONENT SHIFT_REG
      GENERIC (
        pWIDTH: integer := 4
      );
      PORT (
        iCLK, inCLR: IN  STD_LOGIC;
        iLOAD, iSE, iDSL: IN  STD_LOGIC;
        iD: IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
        oQ: OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
      );
    END COMPONENT;

    COMPONENT BROJAC
      GENERIC (
        pWIDTH: integer := 4
      );
      PORT (
        iCLK    : IN  STD_LOGIC;
```

```
      inRESET,
      iEN, iP : IN  STD_LOGIC;
      iDATA   : IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
      oQ      : OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
      oTSE    : OUT STD_LOGIC
    );
  END COMPONENT;

  COMPONENT ADDER
    GENERIC (
      pWIDTH: integer := 4
    );
    PORT (
      iX, iY:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
      iSEL:    IN  STD_LOGIC;
      oZ:      OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
    );
  END COMPONENT;

  COMPONENT CONTROL
    PORT (
      iCLK, inRESET: IN  STD_LOGIC;
      iSTART, iCNT_0, iAY_EQ, iAAC_EQ: IN  STD_LOGIC;
      oLOAD_Y, oLOAD_X, oLOAD_CNT, oLOAD_A, oLOAD_AC,
      oCE, oADD_SUB, oREADY: OUT STD_LOGIC;
      oX_SEL, oA_SEL: OUT STD_LOGIC_VECTOR(1 DOWNTO 0)
    );
  END COMPONENT;

END DELITELJ_PKG;
```

Nakon formiranja pakovanja sa opisom sprega svih komponenti koje se instanciraju prelazi se na opis entiteta projektovanog digitalnog sistema. Projektovani modul delitelja sadrži sledeće ulazno/izlazne signale:

- `iCLK` takt signal,
- `inCLR` signal za brisanje sadržaja svih registara i brojača, aktivan na niskom nivou i sinhron sa signalom takta,
- `iSTART` impuls za početak izvršenja algoritma delenja,
- `iX` deljenik, četvorobitni ulazni vektor,
- `iY` delitelj, četvorobitni ulazni vektor,
- `oZ` količnik, četvorobitni izlazni vektor,
- `oO` ostatak, četvorobitni izlazni vektor i
- `oREADY` izlazni signal koji govori da je u toku izvršenje algoritma, ako je njegova vrednost na niskom logičkom nivou. U suprotnom, `oREADY=1`, sistem je spreman za izvršenje algoritma.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;
  -- ukljucivanje pakovanja sa komponentama
  USE work.DELITELJ_PKG.all;
```

```
ENTITY DELITELJ IS
  PORT (
    iCLK, inCLR: IN  STD_LOGIC;
    iSTART: IN  STD_LOGIC;
    iX, iY: IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    oZ, oO: OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    oREADY: OUT STD_LOGIC );
END DELITELJ;

ARCHITECTURE ARH_DELITELJ OF DELITELJ IS
  -- konstanta nula
  CONSTANT cZERO: STD_LOGIC := '0';
  -- broj bita sa kojima se reprezentuju operandi
  CONSTANT cNUM_BITS: STD_LOGIC_VECTOR(2 DOWNTO 0) := "100";

  -- registar deljenika i delitelja
  SIGNAL sY, sX: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- ulazni vektor u registar za smestanje delitelja
  SIGNAL sDATA: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- akumulator, stanje akumulatora posle pomeranja
  -- i rezultat sabiraca
  SIGNAL sA, sAC, sR: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- ulazni vektor akumulator
  SIGNAL sDATA_A: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- brojac bita
  SIGNAL sCNT: STD_LOGIC_VECTOR(2 DOWNTO 0);
  -- delitelj i akumulator imaju iste znakove
  SIGNAL sAY_EQ: STD_LOGIC;
  -- akumulator i njegova vrednost posle pomeranja
  -- imaju iste znakove
  SIGNAL sAAC_EQ: STD_LOGIC;
  -- upravljacki signali za registre i brojac
  SIGNAL sLOAD_Y, sLOAD_X, sLOAD_A, sLOAD_AC,
         sLOAD_CNT, sCE,
         sCNT_TC, sADD_SUB: STD_LOGIC;
  SIGNAL sX_SEL, sA_SEL: STD_LOGIC_VECTOR(1 DOWNTO 0);

BEGIN

 -- registar za smestanje delitelja
  eREG_Y: REG
    GENERIC MAP (pWIDTH=>4)
    PORT    MAP (iCLK=>iCLK, inCLR=>inCLR, iCE=>sLOAD_Y,
                 iD=>iY, oQ=>sY);

  -- multiplekser za odabir vrednosti
  -- koja se upisuje u registar deljenika
  PROCESS (iX, sX_SEL) BEGIN
    CASE sX_SEL IS
                  -- propusti deljenik
      WHEN "00"   => sDATA <= iX;
                  -- resetuj bit 0
      WHEN "01"   => sDATA <= (sX(3 DOWNTO 1) & '0');
                  -- postavi na 1 bit 0
      WHEN OTHERS => sDATA <= (sX(3 DOWNTO 1) & '1');
    END CASE;
  END PROCESS;
```

```
-- registar za smestanje deljenika
eREG_X: SHIFT_REG
  GENERIC MAP (pWIDTH=>4)
  PORT    MAP (iCLK=>iCLK, inCLR=>inCLR,
               iLOAD=>sLOAD_X, iSE=>sCE, iDSL=>cZERO,
               iD=>sDATA, oQ=>sX);

-- multiplekser za odabir vrednosti
-- koja se upisuje u akumulator
PROCESS (iX, sR, sA_SEL) BEGIN
  CASE sA_SEL IS
    WHEN "00" =>
      -- upisi znak deljenika na sve pozicije
      -- koristi se tokom inicijalizacije delenja
     sDATA_A <= (iX(3) & iX(3) & iX(3) & iX(3));
   WHEN "01" =>
      -- upisi izlaz sabiraca
      -- koristi se tokom operacije delenja
     sDATA_A <= (sR);
   WHEN OTHERS =>
      -- upisi stanje pre pomeranja
      -- koristi se u RESTORE stanju
     sDATA_A <= (sAC);
  END CASE;
END PROCESS;

-- akumulator za smestanje rezultata i izlaznog prenosa
eREG_A: SHIFT_REG
  GENERIC MAP (pWIDTH=>4)
  PORT    MAP (iCLK=>iCLK, inCLR=>inCLR,
               iLOAD=>sLOAD_A, iSE=>sCE, iDSL=>sX(3),
               iD=>sDATA_A, oQ=>sA);

-- registar za smestanje stanja akumulatora posle pomeranja
eREG_AC: REG
  GENERIC MAP (pWIDTH=>4)
  PORT    MAP (iCLK=>iCLK, inCLR=>inCLR, iCE=>sLOAD_AC,
               iD=>sA, oQ=>sAC);

-- sabirac/oduzimac akumulisane vrednosti u
-- akumulatoru i delitelja
-- rezultat se smesta nazad u akumulator
eADDER: ADDER
  GENERIC MAP (pWIDTH=>4)
  PORT MAP (iX=>sA, iY=>sY, iSEL=>sADD_SUB, oZ=>sR);

-- brojac ciklusa (broja obradjenih bita) tokom delenja
eCNT: BROJAC
  GENERIC MAP (pWIDTH=>3)
  PORT    MAP (iCLK=>iCLK, inRESET=>inCLR, iEN=>sCE,
               iP=>sLOAD_CNT, iDATA=>cNUM_BITS,
               oQ=>sCNT, oTSE=>sCNT_TC);

-- =1 ako akumulator i delitelj imaju iste znakove
sAY_EQ  <= sA(3) XNOR sY(3);
-- =1 ako akumulator i njegova vrednost
-- pre pomeranja imaju iste znakove
sAAC_EQ <= sA(3) XNOR sAC(3);
```

```
    -- automat ze generisanje upravljackih signala
    eCONTROL: CONTROL
      PORT MAP (iCLK=>iCLK, inRESET=>inCLR,
                iSTART=>iSTART, iCNT_0=>sCNT_TC,
                iAY_EQ=>sAY_EQ, iAAC_EQ=>sAAC_EQ,
                oLOAD_Y=>sLOAD_Y, oLOAD_X=>sLOAD_X,
                oLOAD_CNT=>sLOAD_CNT, oLOAD_A=>sLOAD_A,
                oLOAD_AC=>sLOAD_AC, oCE=>sCE, oA_SEL=>sA_SEL,
                oX_SEL=>sX_SEL, oADD_SUB=>sADD_SUB, oREADY=>oREADY);

    -- proces za odredjivanje vrednosti rezultata
    PROCESS (iX, iY, sX) BEGIN
      IF (iX(3) = iY(3)) THEN
        -- deljenik i delitelj imaju iste znakove
        oZ <= sX; -- rezultat
      ELSE
        -- deljenik i delitelj nemaju iste znakove
        -- napravi II komplement dobijenog rezultata
        oZ <= STD_LOGIC_VECTOR(UNSIGNED(NOT(sX)) + "0001");
      END IF;
    END PROCESS;

    oO <= sA; -- ostatak

  END ARH_DELITELJ;
```

Za registar X pored operacije pomeranja u levo, treba da se omogući upis vrednosti deljenika, postavljanje bita X_0 na vrednost nula ili na vrednost 1. Pomeranje sadržaja registra u levo je realizovano instanciranjem odgovarajućeg pomeračkog registra i pravovremenim generisanjem signala dozvole pomeranja. Upis delitelja i postavljanje odgovarajuće vrednosti bita najniže važnosti je realizovano uvođenjem multipleksera na isti način kao u prethodnom zadatku.

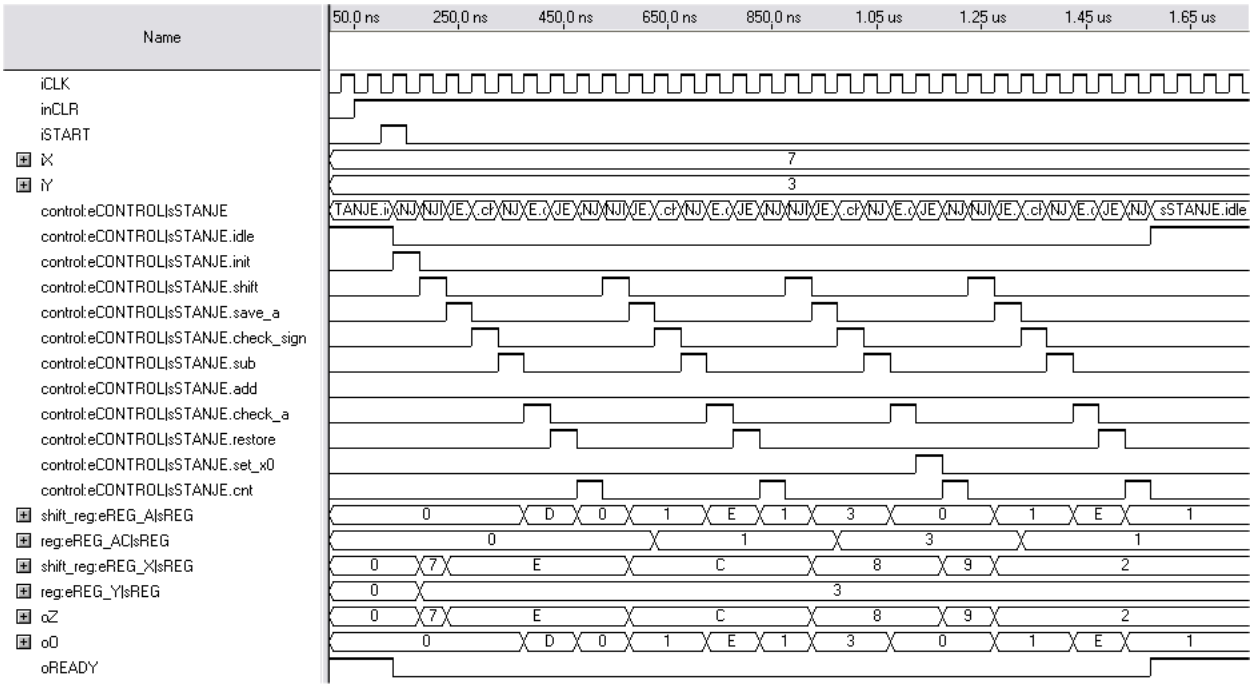
Akumulatorski registar A, kao što je ranije navedeno, može da primi ukupno tri vrednosti: inicijalizacionu vrednost gde svi biti primaju znak deljenika, rezultat sabirača/oduzimača i obnavljanje sadržaja akumulatora sa vrednošću iz registra AC. To je realizovano uvođenjem multipleksera čiji se izlaz, signal `sDATA_A`, upisuje u akumulatorski registar preko ulaznog vektora `iD`:

```
    PROCESS (iX, sR, sA_SEL) BEGIN
      CASE sA_SEL IS
        WHEN "00" =>
          -- upisi znak deljenika na sve pozicije
          -- koristi se tokom inicijalizacije delenja
         sDATA_A <= (iX(3) & iX(3) & iX(3) & iX(3));
       WHEN "01" =>
          -- upisi izlaz sabiraca
          -- koristi se tokom operacije delenja
         sDATA_A <= (sR);
       WHEN OTHERS =>
          -- upisi stanje pre pomeranja
          -- koristi se u RESTORE stanju
         sDATA_A <= (sAC);
      END CASE;
    END PROCESS;
```

U slučaju da upravljački signal sA_SEL ima vrednost "00" generisanjem signala dozvole upisa sLOAD_A u izvršava sa inicijalizacija sadržaja akumulatora. Kada signal sA_SEL ima vrednost "01" u akumulator se upisuje izlazna vrednost sabirača/oduzimača koju predstavlja signal sR. Za vrednost "10" signala sA_SEL izvršava se obnavljanje sadržaja akumulatora smeštenog u registru AC (signal sAC).

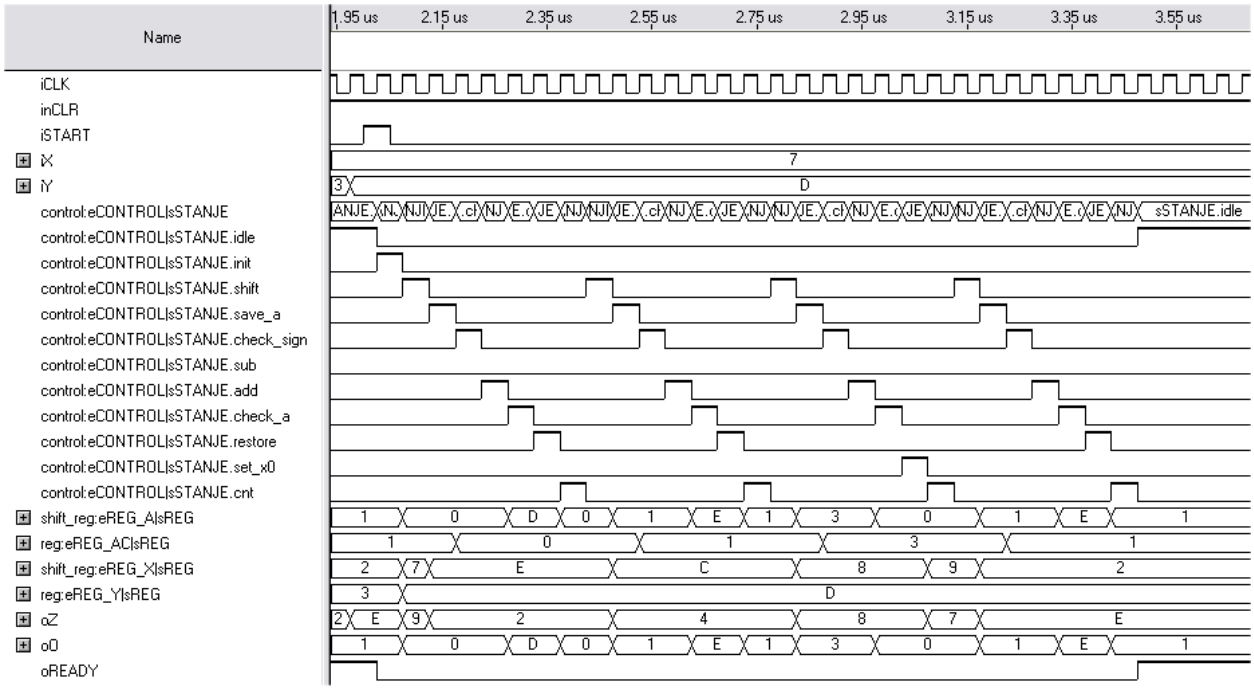
Dobijeni količnik, sX, se prosleđuje na izlazni vektor oZ u slučaju da deljenik iX i delitelj iY imaju iste znakove. U suprotnom slučaju, znakovi su različiti, na izlaz se prosleđuje drugi komplement od dobijenog količnika.

Situaciju kada deljenik i delitelj imaju iste znakove prikazuje Slika 8.45. Slika prikazuje vremenski dijagram delenja 7 sa 3, gde se za rezultat dobija količnik 2 (oZ) i ostatak 1 (oO). Vidi se da je vrednost izlaznog vektora oZ jednaka sadržaju registra X (REG_X).



Slika 8.45: Simulacija rada realizovanog delitelja za delenje 7 sa 3

Sa druge strane, vremenski dijagram izvršenja operacije delenja u slučaju da deljenik i delitelj imaju različite znakove prikazuje Slika 8.46. Prikazan je slučaj delenja 7 sa –3. Rezultat delenja je količnik –2 i ostatak 1. Vrednost –2 predstavljena heksadecimalno u drugom komplementu je E. Upravo ova vrednost je prisutna na izlaznom vektoru oZ, dok je u registru REG_X prisutna vrednost 2 (apsolutna vrednost količnika).



Slika 8.46: Simulacija rada realizovanog delitelja za delenje 7 sa –3

PROJEKTOVANJE UPRAVLJAČKE JEDINICE PROCESORA

8.12 ZADATAK:

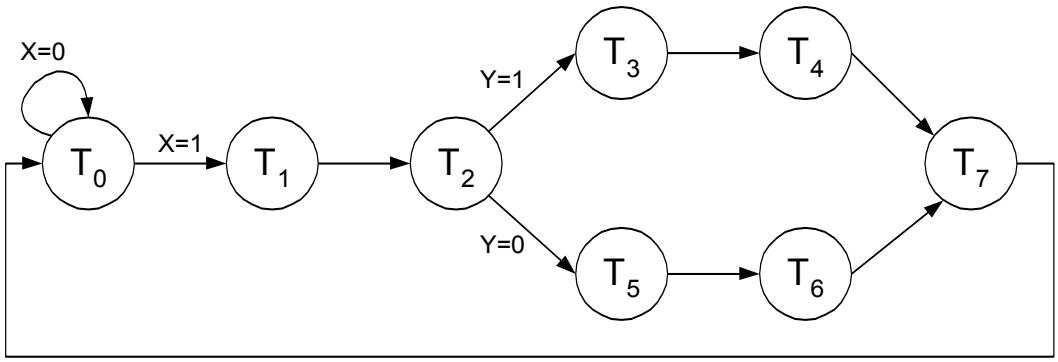
Projektovati UJ sa dva ulazna signala (x i y), i osam stanja koju prikazuje Slika 8.47. Realizaciju izvršiti:

a) Metodom dodele jednom stanju jednog memorijskog elementa (D flip-flopovi i standardna logička kola).
b) Pomoću dekodera i registra sekvence. Registar sekvence realizovati sa JK flip-flopovima.
c) Pomoću PLA i registra sekvence. Registar sekvence realizovati sa JK flip-flopovima.

Qn	Qn+1	J	K
0	0	0	×
0	1	1	×
1	0	×	1
1	1	×	0

Tabela 8.12: Funkcionalna tabela JK flip-flopa

Početno stanje upravljačke jedinice je stanje T_0.



Slika 8.47: Graf UJ koju treba realizovati

REŠENJE:

a).

Projektovanje UJ započinje definisanjem ulaznih signala i funkcije prelaza. Izlazni signali se definišu na osnovu strukture ALJ potrebne za rešenje problema.

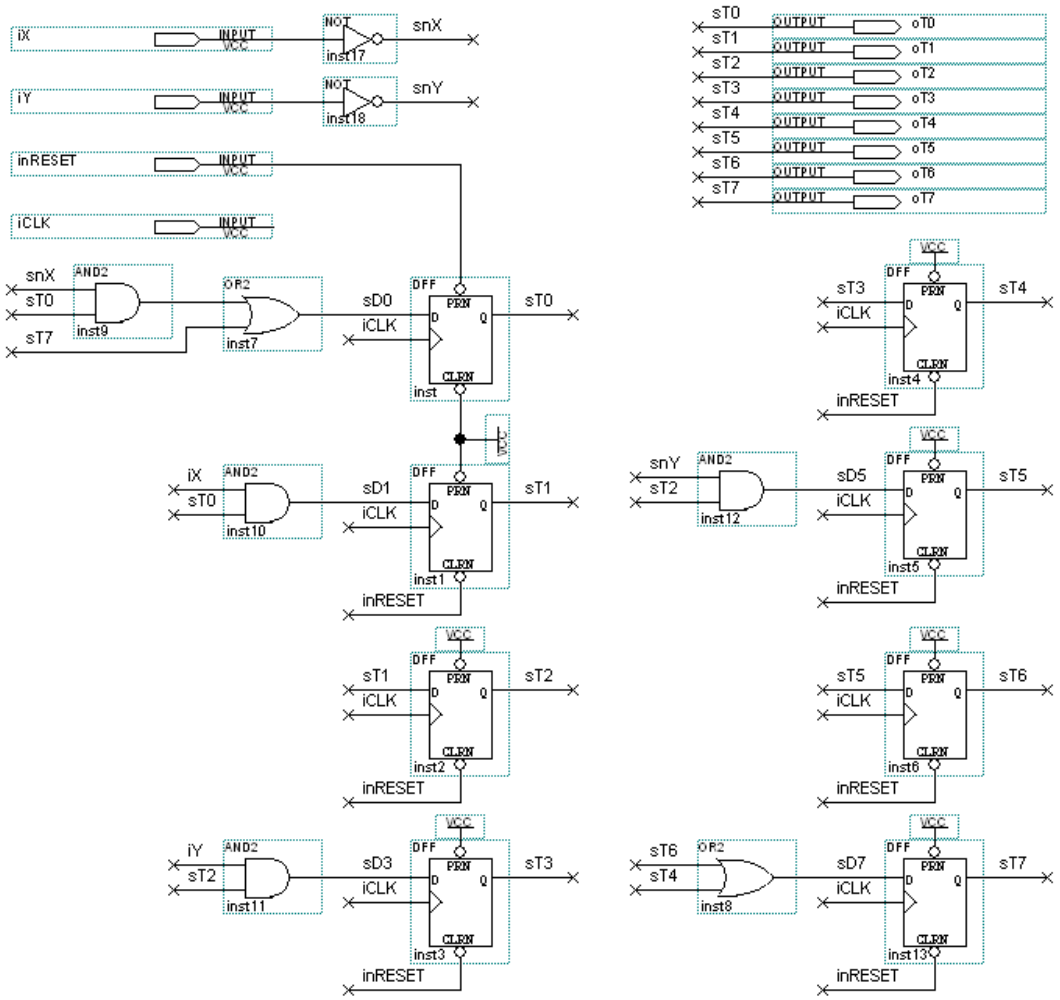
Kako u realizaciji UJ ovom metodom na raspolaganju stoje D flip flopovi moguće je direktno definisanje ulaza u flip flopove. Posmatranjem grafa (Slika 8.47) vidi se da se iz stanja T_0 prelazi u stanje T_0 ako je X=0, tj. preko puta $\overline{X}$, a iz T_0 u T_1 preko puta X. Ako se za svako stanje T_0 - T_7 na ovaj način definišu ulazi, dobijaju se funkcije ulaza u D flip flopove:

$$D_{T0} = \overline{X} \cdot T_0 + T_7$$
$$D_{T1} = X \cdot T_0$$
$$D_{T2} = T_1$$
$$D_{T3} = Y \cdot T_2$$
$$D_{T4} = T_3$$
$$D_{T5} = \overline{Y} \cdot T_2$$
$$D_{T6} = T_5$$
$$D_{T7} = T_6 + T_4$$

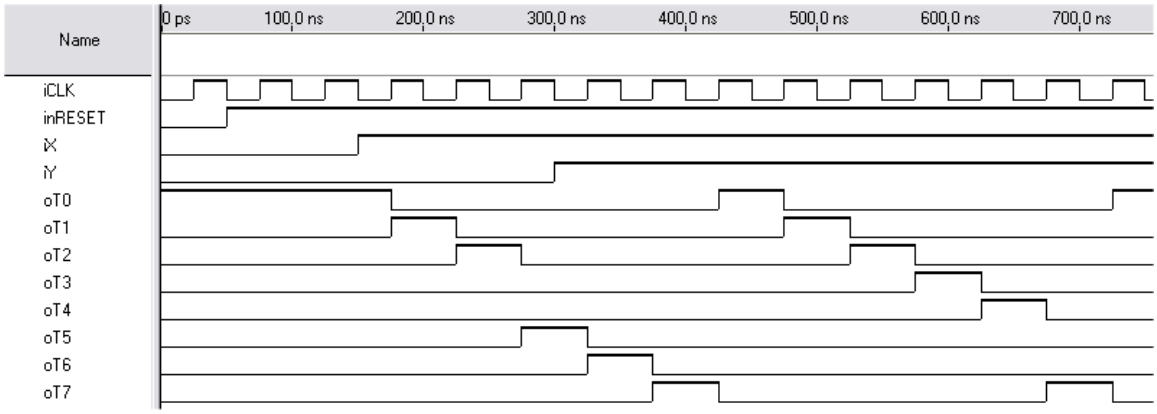
Na osnovu funkcija ulaza u flip flopove može se projektovati logička šema tražene upravljačke jedinice (Slika 8.48). Izlazi iz upravljačke jedinice su signali T_0, T_1, … , T_7. Na osnovu tih signala se može rekonstruisati u kom stanju se trenutno nalazi realizovana upravljačka jedinica.

Slika 8.49 prikazuje simulaciju rada realizovane upravljačke jedinice. Na početku vremenskog dijagrama upravljačka jedinica je postavljena u početno stanje aktiviranjem ulaznog signala `inRESET`. Nakon toga, oba ulazna signala su na niskom nivou, pa prema zadatom dijagramu prelazaka stanja upravljačka jedinica ostaje u stanju T_0. Prelaskom ulaznog signala `iX` na visok nivo, upravljačka jedinica prelazi u stanje T_1 i odmah zatim u stanje T_2. U stanju T_2 se analizira ulazni signal `iY`. Pošto dati signal ima vrednost nula sledeće stanje je stanje T_5. Zatim sledi sekvenca stanja T_6, T_7 i T_0. Pošto je vrednost ulaznog signala `iX`

jednaka jedinici u datom trenutku, upravljačka jedinica prelazi u stanje T_1 i sa sledećom rastućom ivicom takt signala `iCLK` prelazi se u stanje T_2. Utvrđivanjem vrednosti ulaznog signala iY upravljačka jednica prelazi u stanje T_3 pošto je dati signal na visokom nivou. Nakon toga sledi sekvenca stanja T_4, T_7 i T_0. Time su simulirani svi prelasci stanja koje sadrži zadata upravljačka jedinica.



Slika 8.48: Logička šema upravljačke jedinice realizovane po principu jedno stanje - jedan memorijski element



Slika 8.49: Simulacija rada UJ

b).

Postupak projektovanja ove UJ podrazumeva prvo kodiranje stanja UJ a zatim crtanje programske tablice(Tabela 8.13). Neka je kodiranje stanja izvršeno na sledeći način:

$T_0 \leftarrow 000$ $T_1 \leftarrow 001$ $T_2 \leftarrow 010$ $T_3 \leftarrow 011$
$T_4 \leftarrow 100$ T5$\leftarrow$101 $T_6 \leftarrow 110$ $T_7 \leftarrow 111$

UJ sa osam stanja realizuje se jednim registrom dužine 3 bita i dekoderom 3×8.Ako je registar realizovan JK flip-flopovima (Tabela 8.12) moguće je definisati tabelu UJ, čije kolone čine trenutno stanje registra (G_3 G_2 G_1), ulazi (X, Y), naredno stanje registra (G_3' G_2' G_1'), funkcije ulaza u flip-flopove i oznaka stanja UJ tj. izlazi dekodera.

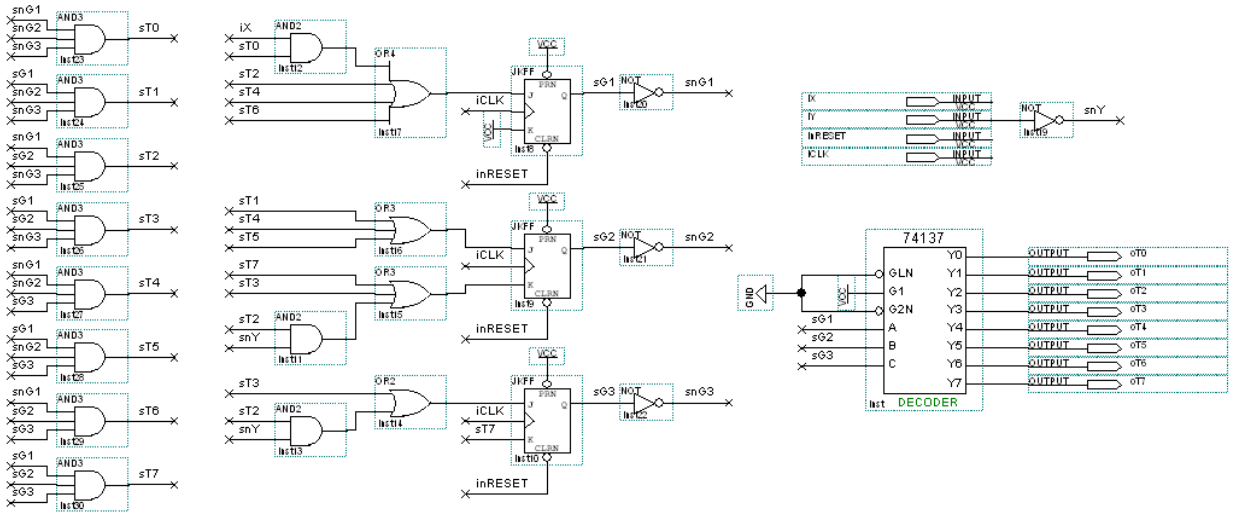
Ti	$G_3G_2G_1$	X	Y	$G_3'G_2'G_1'$	J_3	K_3	J_2	K_2	J_1	K_1
T_0	000	0	×	000	0	×	0	×	0	×
T_0	000	1	×	001	0	×	0	×	1	×
T_1	001	×	×	010	0	×	1	×	×	1
T_2	010	×	1	011	0	×	×	0	1	×
T_2	010	×	0	101	1	×	×	1	1	×
T_3	011	×	×	100	1	×	×	1	×	1
T_4	100	×	×	111	×	0	1	×	1	×
T_5	101	×	×	110	×	0	1	×	×	1
T_6	110	×	×	111	×	0	×	0	1	×
T_7	111	×	×	000	×	1	×	1	×	1

Tabela 8.13: Programska tablica UJ

Na osnovu tabele 1 moguće je definisati funkcije za ulaze flip-flopova:

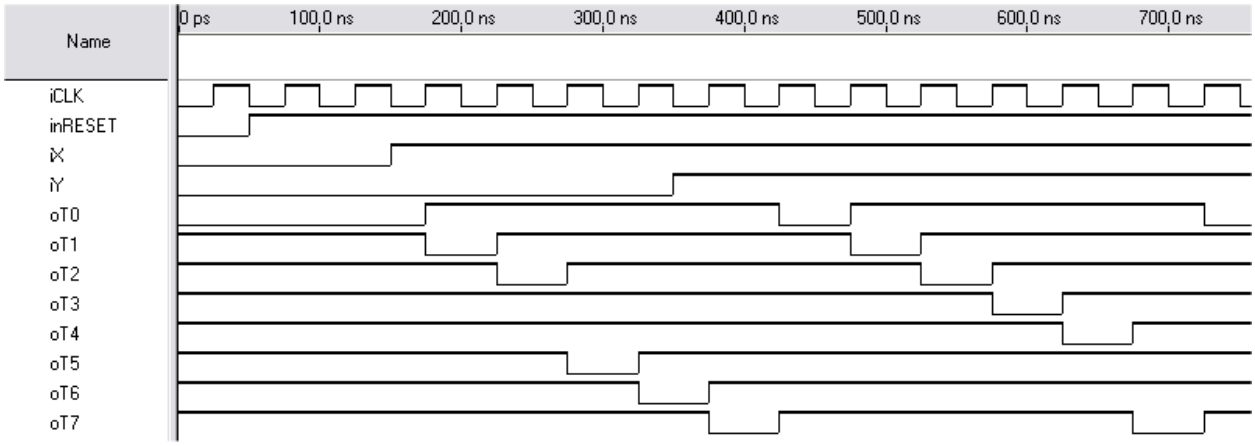
$$J_3 = T_2 \cdot \overline{Y} + T_3 \qquad K_3 = T_7$$
$$J_2 = T_1 + T_4 + T_5 \qquad K_2 = T_2 \cdot \overline{Y} + T_3 + T_7$$
$$J_1 = X \cdot T_0 + T_2 + T_4 + T_6 \qquad K_1 = 1$$

Na osnovu dobijenih jednačina pristupa se sintezi logičke šeme upravljačke jedinice, Slika 8.50. Na šemi je za dekoder iskorišćena komponenta 74137. Ova komponenta predstavlja dekoder 3×8 sa tri dodatna ulaza dozvole rada dekodera; `G1`, `G2N` i `GLN`. Da bi dekoder na izlazima `Y0` do `Y7` postavio dekodovanu vrednost prisutnu na ulaznim signalima `C`, `B` i `A` (`C` je bit najveće važnosti dok je `A` bit najmanje važnosti) potrebno je ulaz `G1` postaviti na visok logički nivo, a ulaze `G2N` i `GLN` na nizak logički nivo. Vrednost na izlazu kod ovog dekodera je u invertovanoj logici.



Slika 8.50: Logička šema UJ realizovane sa dekoderom i registrom sekvence

Slika 8.51 prikazuje vremenski dijagram simulacije rada realizovane upravljačke jedinice. Prikazani prelasci stanja su identični kao i kod upravljačke jedinice realizovane metodom jedan memorijski element po stanju. Jedina razlika je u tome što su izlazni signali u ovom slučaju u invertovanoj logici zbog upotrebljenog dekodera.



Slika 8.51: Vremenski dijagram rada UJ

c).

U projektovanju UJ na bazi PLA neophodno je definisanje programske tablice. Prva kolona programske tablice predstavlja indeks elementarnog proizvoda uz koje slede kolona ulaza, izlaza i kolona komentara. Izlazne promenljive su i promenljive stanja.

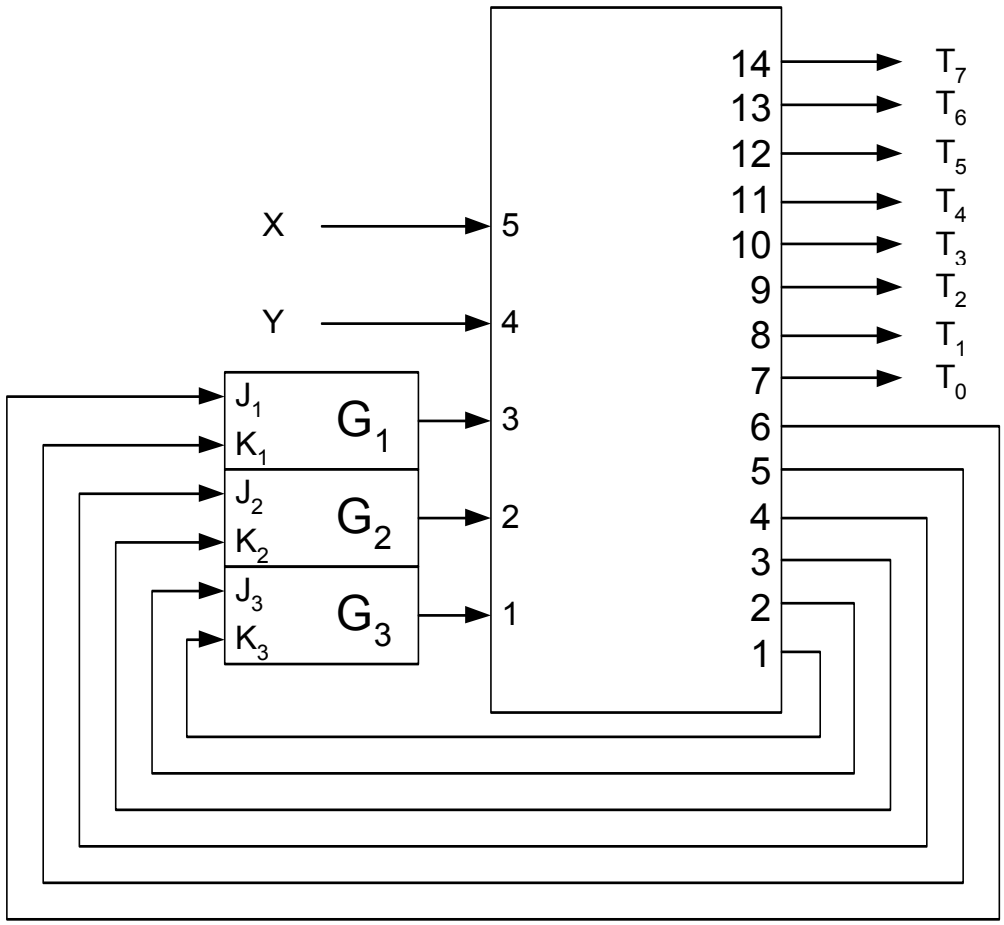
Neka je registar sekvence realizovan pomoću JK flip-flopova (Tabela 8.12) pa se pre definisanja programske tablice za PLA može definisati Tabela 8.14.

$G_3G_2G_1$	X	Y	$G_3'G_2'G_1'$	J_3	K_3	J_2	K_2	J_1	K_1	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7
000	0	×	000	0	×	0	×	0	×	1	0	0	0	0	0	0	0
000	1	×	001	0	×	0	×	1	×	1	0	0	0	0	0	0	0
001	×	×	010	0	×	1	×	×	1	0	1	0	0	0	0	0	0
010	×	1	011	0	×	×	0	1	×	0	0	1	0	0	0	0	0
010	×	0	101	1	×	×	1	1	×	0	0	1	0	0	0	0	0
011	×	×	100	1	×	×	1	×	1	0	0	0	1	0	0	0	0
100	×	×	111	×	0	1	×	1	×	0	0	0	0	1	0	0	0
101	×	×	110	×	0	1	×	×	1	0	0	0	0	0	1	0	0
110	×	×	111	×	0	×	0	1	×	0	0	0	0	0	0	1	0
111	×	×	000	×	1	×	1	×	1	0	0	0	0	0	0	0	1

Tabela 8.14: Tabela ulaza i stanja UJ

Na osnovu prethodne tabele dobijaju se jednačine ulaza u flip-flopove koje su identične kao i kod rešenja pod b).

Vidi se da korišćena PLA treba da ima 5 ulaza (X, Y, G_3, G_2, G_1), 14 izlaza (J_3, K_3, J_2, K_2, J_1, K_1, T_0, … , T_7) i 10 proizvoda. Slika 8.52 prikazuje blok šemu ove UJ.



Slika 8.52: Blok šema UJ realizovane pomoću PLA

Sada je moguće definisati programsku tablicu za PLA (Tabela 8.15). Prilikom kreiranja programske tablice koriste se sledeće oznake:

- "1" u I matrici znači da se ulazni signal u određenom proizvodu ne komplementira, "0" označava komplementiran ulaz, a "-" ulaz koji ne učestvuje u konkretnom proizvodu.
- "1" u ILI matrici označava da proizvod učestvuje u formiranju disjunktivne forme, a "-" je oznaka za proizvod koji ne učestvuje u disjunkciji.

pr.	ulazi (I matrica)					izlazi (ILI matrica)													
	1 G_1	2 G_2	3 G_3	4 X	5 Y	1 J_3	2 K_3	3 J_2	4 K_2	5 J_1	6 K_1	7 T_0	8 T_1	9 T_2	10 T_3	11 T_4	12 T_5	13 T_6	14 T_7
1	0	0	0	0	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-
2	0	0	0	1	-	-	-	-	-	1	-	1	-	-	-	-	-	-	-
3	0	0	1	-	-	-	-	1	-	-	1	-	1	-	-	-	-	-	-
4	0	1	0	-	1	-	-	-	-	1	-	-	-	1	-	-	-	-	-
5	0	1	0	-	0	1	-	-	1	1	-	-	-	1	-	-	-	-	-
6	0	1	1	-	-	1	-	-	1	-	1	-	-	-	1	-	-	-	-
7	1	0	0	-	-	-	-	1	-	1	-	-	-	-	-	1	-	-	-
8	1	0	1	-	-	-	-	1	-	-	1	-	-	-	-	-	1	-	-
9	1	1	0	-	-	-	-	-	-	1	-	-	-	-	-	-	-	1	-
10	1	1	1	-	-	-	1	-	1	-	1	-	-	-	-	-	-	-	1

Tabela 8.15: Programska tablica PLA

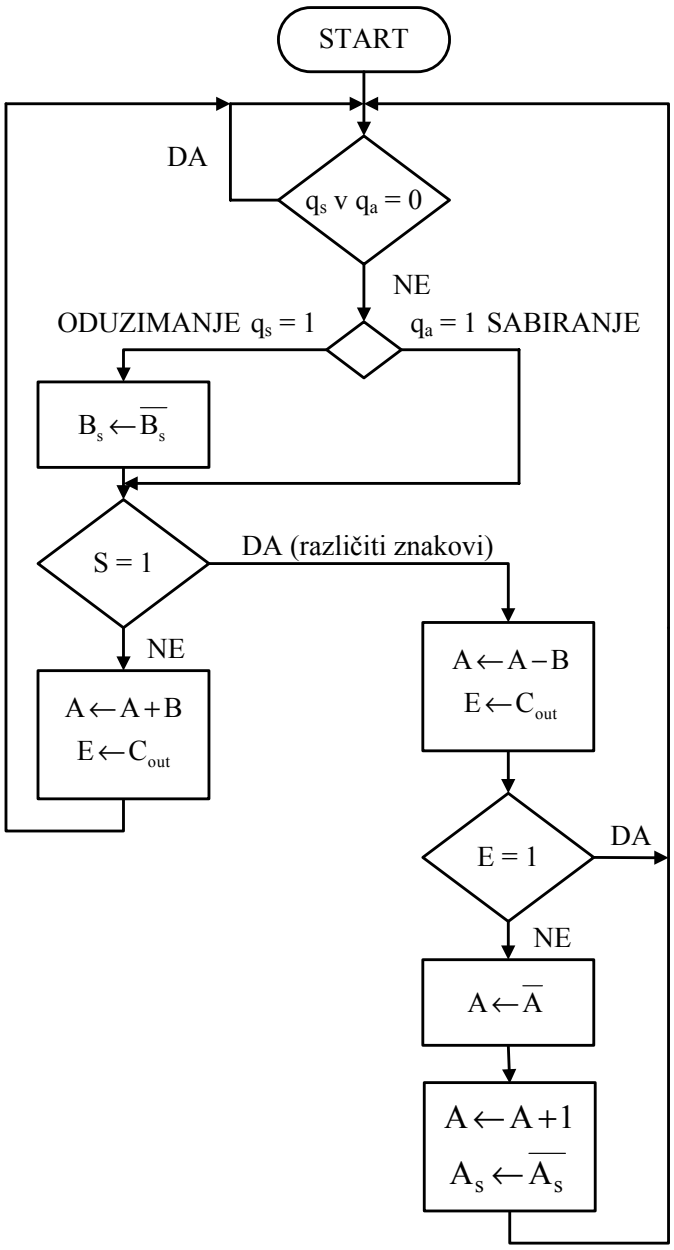
8.13 ZADATAK:

Slika 8.53 prikazuje blok dijagram algoritma za sabiranje označenih brojeva. Skicirati blok šemu digitalnog sistema za izvršenje datog algoritma.
Nakon toga, pomoću VHDL jezika za opis fizičke arhitekture, projektovati UJ za sabiranje označenih brojeva prema zadatom algoritmu metodom jedan memorijski element po stanju. Takođe, izvršiti sintezu ekvivalentnog automata sa kodovanim stanjima koji izvršava isti algoritam.

Izvršiti poređenje zauzetih resursa u programibilnoj mreži ALTERA FLEX 10K30 kao i poređenje maksimalne radne frekvencije oba rešenja.

REŠENJE:

Iz polaznog dijagrama vidi se da su ulazni signali u upravljački automat q_a, q_s, S i E. Izlazni signali se definišu polazeći od strukture ALJ za rešavanje problema obrade.



Slika 8.53: Polazni dijagram toka za sabiranje označenih brojeva

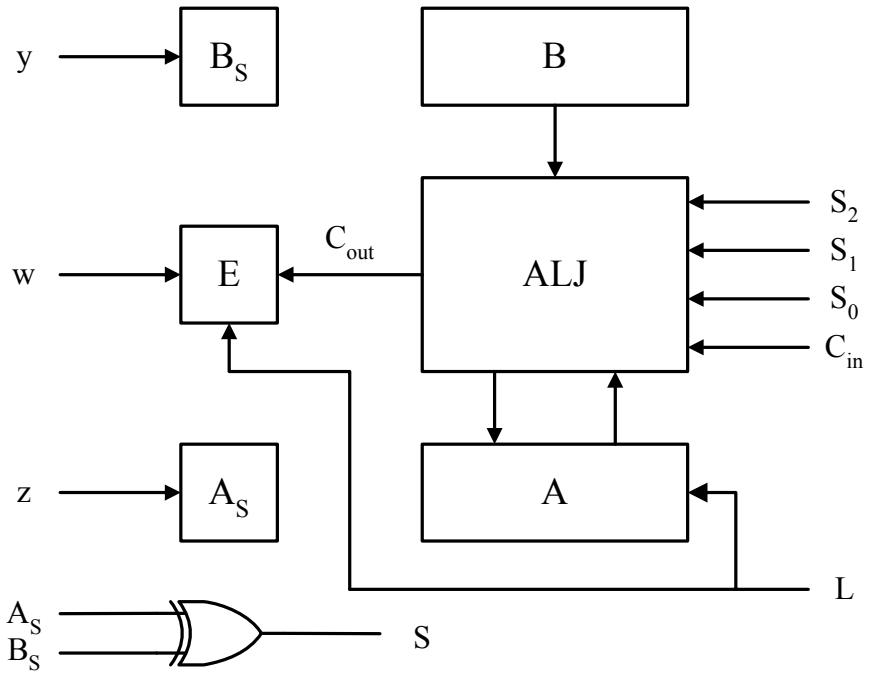
U polaznom dijagramu toka (Slika 8.53) uočava se da su za realizaciju mikrooperacija neophodne sledeće komponente:

- Za početak rada jedinice neophodno je obezbediti signal X koji služi za uključenje.
- Registri A i B služe za čuvanje operanada koji učestvuju u aritmetičkim operacijama. Pri tome, veza između ALJ i registra A je dvosmerna, jer se rezultat operacije smešta u registar A, koji se naziva još i AKUMULATOR.
- Memorijski elementi za čuvanje znaka brojeva A_S i B_S. Za komplementiranje njihovih vrednosti treba dovesti signale upisa u memorijske elemente. Označićemo ih sa y za komplement znaka B_S i z za komplement znaka A_S.

- Signal S koji ukazuje da li su znaci operanda isti ili različiti ($S = A_S \oplus B_S$)
- Memorijski element za prekoračenje iz ALJ (E) koji se briše signalom upisa nule w
- Aritmetičko logička jedinica, ALJ. Na primer, neka se koristi ALJ koja je realizovana u poslednjem zadatku u poglavlju o projektovanju ALJ. Ova ALJ ima tri selekciona ulaza, S_2, S_1, S_0, i ulazni prenos C_{in}, a operacije koje ona izvodi prikazuje Tabela 8.16.

S_2	S_1	S_0	C_0	Izlaz	Funkcija
0	0	0	0	F = A	prenos
0	0	0	1	F = A + 1	uvećanje A
0	0	1	0	F = A + B	sabiranje
0	0	1	1	F = A + B + 1	sabiranje
0	1	0	0	F = A – B - 1	oduzimanje
0	1	0	1	F = A - B	oduzimanje
0	1	1	0	F = A - 1	umanjenje A
0	1	1	1	F = A	prenos
1	0	0	0	$F = A \vee B$	ILI
1	0	1	0	$F = A \oplus B$	XILI
1	1	0	0	$F = A \wedge B$	I
1	1	1	0	$F = \overline{A}$	NE

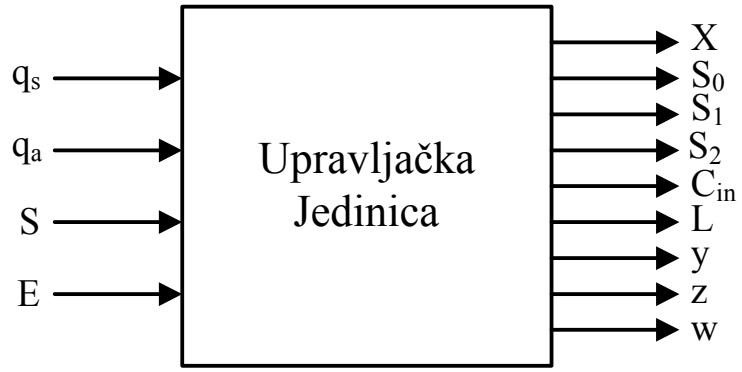
Tabela 8.16: Operacije ALJ u zavisnosti od selekcionih promenljivih



Slika 8.54: Blok šema mikroprocesora za sabiranje označenih brojeva, čijim radom upravljaja UJ

Slika 8.54 prikazuje mikroprocesor sa komponentama potrebnim za realizaciju traženog algoritma, kao i sa signalima koje generiše upravljačka jedinica.

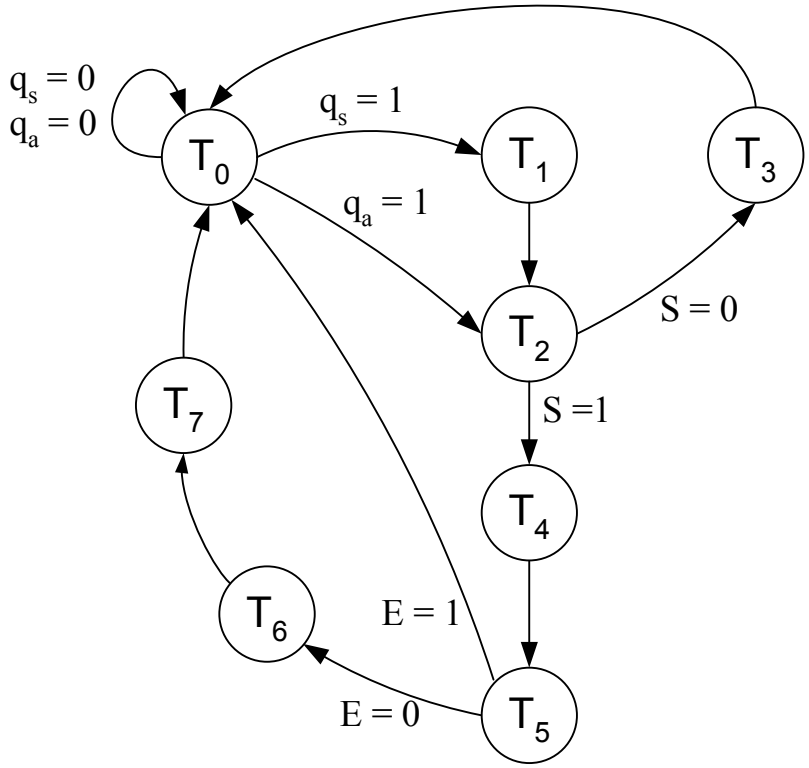
Sa oznakom L je označen signal za dozvolu upisa vrednosti u akumulator i flip-flop za memorisanje prekoračenja.Ulazno-izlazne signale upravljačke jedinice prikazuje Slika 8.55.



Slika 8.55: Struktura UJ za upravljanje ALJ koja realizuje sabiranje označenih brojeva

Iz dijagrama algoritma se vidi da ima 8 operatora kojima se dodeljuje 8 stanja T_0 do T_7 , odnosno operatori se izvršavaju pojedinačno.

Na osnovu blok dijagrama algoritma moguće je nacrtati graf upravljačke jedinice (Slika 8.56).



Slika 8.56: Graf UJ

Može se formirati i tablica za operatore T_0 do T_7 kao vrste u kojima su navedene mikrooperacije koje se izvršavaju kao i upravljački signali koji su u tom momentu aktivni. To prikazuje Tabela 8.17.

		Izlazni signali								
		X	S_2	S_1	S_0	C_{in}	L	y	z	w
T_0	početno stanje	1	0	0	0	0	0	0	0	0
T_1	$B_S \leftarrow \overline{B_S}$	0	0	0	0	0	0	1	0	0
T_2	provera S	0	0	0	0	0	0	0	0	0
T_3	$A \leftarrow A + B$, $E \leftarrow C_{out}$	0	0	0	1	0	1	0	0	0
T_4	$A \leftarrow A + \overline{B} + 1$ $E \leftarrow C_{out}$	0	0	1	0	1	1	0	0	0
T_5	provera E	0	0	0	0	0	0	0	0	1
T_6	$A \leftarrow \overline{A}$	0	1	1	1	0	1	0	0	0
T_7	$A \leftarrow A + 1$ $A_S \leftarrow \overline{A_S}$	0	0	0	0	1	1	0	1	0

Tabela 8.17: Tabela izlaznih signala UJ

Tabela 8.17 definiše funkcije izlaza upravljačkog automata kao:

$X = T_0$ $\quad L = T_3 + T_4 + T_6 + T_7$

$S_2 = T_6$ $\quad y = T_1$

$S_1 = T_4 + T_6$ $\quad z = T_7$

$S_0 = T_3 + T_6$ $\quad w = T_5$

$C_{ul} = T_4 + T_7$

Posmatranjem grafa prelaza stanja upravljačke jedinice (Slika 8.53), vidi se da se u stanje T_0 prelazi iz stanja T_0 preko puta $\overline{q_s} \cdot \overline{q_a}$, iz stanja T_3, iz stanja T_5 uz uslov E=1, iz stanja T_7. Slično, u stanje T_1 se prelazi iz T_0 uz uslov q_s=1, u stanje T_2 iz T_1 i iz T_0 pod uslovom da je q_a=1, i tako redom. Imajući u vidu osobinu D flip-flopova da prenose informaciju prisutnu na ulazu, disjunkcijom puteva iz kojih se prelazi u neko naredno stanje, dobija se jednačina pobude svakog D flip-flopa koji je dodeljen stanju. Prema tome, uz pretpostavku da se izlazi D flip-flopova označe sa T_I, i=0, 1 ,..., 7 imaćemo:

$D_0 = T_7 + T_5 \cdot E + T_3 + T_0 \cdot \overline{q_s} \cdot \overline{q_a}$ $\quad D_4 = T_2 \cdot S$

$D_1 = T_0 \cdot q_s$ $\quad D_5 = T_4$

$D_2 = T_1 + T_0 \cdot q_a$ $\quad D_6 = T_5 \cdot \overline{E}$

$D_3 = T_2 \cdot \overline{S}$ $\quad D_7 = T_6$

UJ radi tako što je u jednom trenutku vremena samo jedan flip-flop aktivan, a svi ostali imaju nulti izlaz. Sledeća rastuća ivica takta aktivira samo flip-flop na čijem je D ulazu 1, svi ostali se brišu. Drugim rečima ulazne funkcije flip-flopova su međusobno isključive i samo jedan flip-flop može biti aktivan. Prelaz u naredno stanje iz tekućeg je funkcija tekućeg T_i , koje je 1, i nekih ulaznih uslova. Naredno stanje počinje po brisanju prethodnog. Da bi ovaj upravljački automat mogao da počne da radi, mora se inicirati početno stanje (T_0) i to tako što će se flip-flop koji reprezentuje to stanje staviti u aktivno stanje.

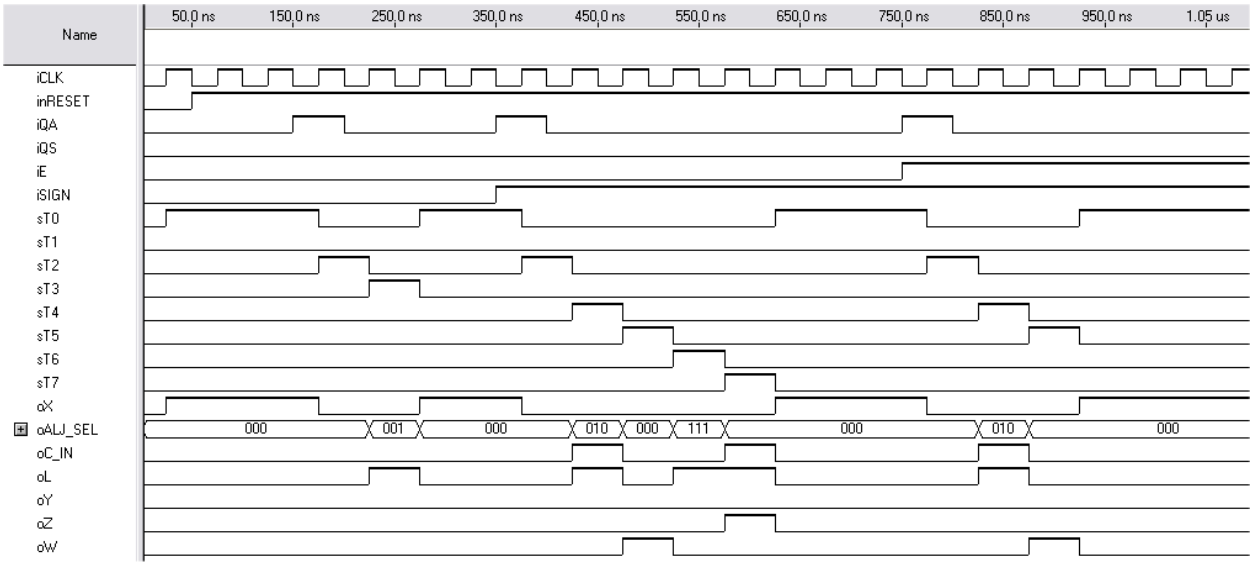
Na osnovu ovih informacija može se pristupiti sintezi VHDL koda realizacije upravljačke jedinice metodom 1 memorijski element po stanju. Arhitektura upravljačke jedinice se sastoji iz tri dela. Prvi deo predstavlja kombinaciona mreža koja generiše pobudu flip-flopova, odnosno kombinaciona mreža koja generiše naredno stanje upravljačke jednice. Ovi signali su u VHDL kodu označeni literalima `sD0`, `sD1`, … , `sD7`. Drugi deo predstavljaju sami flip-flopovi. Postavljanje početnog stanja je realizovano sinhronim reset signalom `inRESET` aktivnim na niskom nivou. Izlazi flip-flopova su u VHDL označeni kao `sT0`, `sT1`, … , `sT7`. Treći deo predstavlja dekodersku logiku koja na osnovu stanja flip-flopova, tj. stanja upravljačke jedinice, generiše odgovarajuće upravljačke signale.

Odgovarajući VHDL kod je prikazan u nastavku.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY UJ2 IS PORT(
  iCLK, inRESET,
  iE, iSIGN,iQA, iQS:         IN  STD_LOGIC;
  oX, oC_IN, oL, oY, oZ, oW: OUT STD_LOGIC;
  oALJ_SEL:                   OUT STD_LOGIC_VECTOR(2 DOWNTO 0) );
END UJ2;

ARCHITECTURE ARH_UJ2 of UJ2 IS
  -- deklaracija signala koji odredjuju tekuce stanje UJ
  -- (izlazi flip-flopova)
  SIGNAL sT0, sT1, sT2, sT3, sT4, sT5, sT6, sT7 : STD_LOGIC;
  -- deklaracija signala koji odredjuju naredno stanje UJ
  -- (ulazi flip-flopova)
  SIGNAL sD0, sD1, sD2, sD3, sD4, sD5, sD6, sD7 : STD_LOGIC;
BEGIN

  -- kombinaciona mreza koja na osnovu trenutnog stanja
  -- i vrednosti ulaza generise kod narednog stanja
  sD0 <= sT7 OR (sT5 AND iE) OR sT3 OR
         (sT0 AND NOT (iQS) AND NOT(iQA));
  sD1 <= sT0 AND iQS;
  sD2 <= sT1 OR (sT0 AND iQA);
  sD3 <= sT2 AND NOT (iSIGN);
  sD4 <= sT2 AND iSIGN;
  sD5 <= sT4;
  sD6 <= sT5 AND NOT(iE);
  sD7 <= sT6;
```

```
    -- flip-flopovi za smestanje koda
    -- trenutnog stanja; postavljanje pocetnog
    -- stanja je sinhrono sa signalom takta
    PROCESS (iCLK) BEGIN
      IF (iCLK'event and iCLK='1') THEN
        IF (inRESET = '0') then  -- sinhroni reset
          sT0 <= '1';  sT1 <= '0';
          sT2 <= '0';  sT3 <= '0';
          sT4 <= '0';  sT5 <= '0';
          sT6 <= '0';  sT7 <= '0';
        ELSE
          sT0 <= sD0;  sT1 <= sD1;
          sT2 <= sD2;  sT3 <= sD3;
          sT4 <= sD4;  sT5 <= sD5;
          sT6 <= sD6;  sT7 <= sD7;
        END IF;
      END IF;
    END PROCESS;

     -- odredjivanje vrednosti izlaznih signala
     -- na osnovu trenutnog stanja
     oX       <= sT0;
     oALJ_SEL <= (sT6 & (sT4 OR sT6) & (sT3 OR sT6));
     oC_IN    <= sT4 OR sT7;
     oL       <= sT3 OR sT4 OR sT6 OR sT7;
     oY       <= sT1;
     oZ       <= sT7;
     oW       <= sT5;

END ARH_UJ2;
```

Slika 8.57 prikazuje vremenski dijagram sabiranja brojeva. Prikazane su sve tri putanje kroz koje može da prođe upravljačka jedinica prilikom izvršenja operacije sabiranja brojeva.



Slika 8.57: Vremenski dijagram sabiranja brojeva

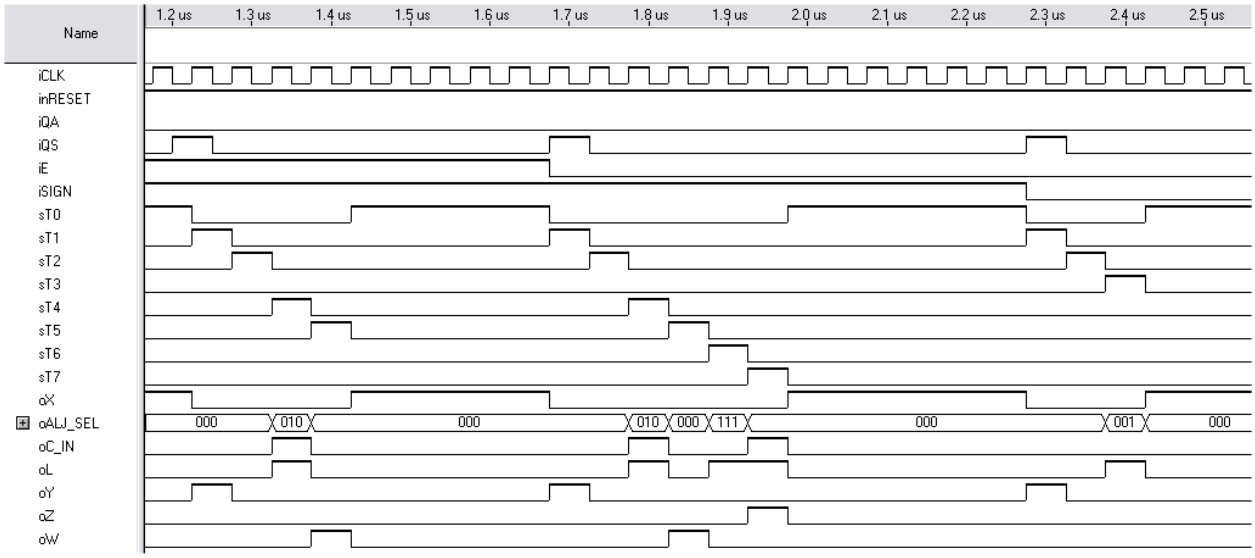
Prvo je prikazana situacija kada operandi imaju iste znakove, tj. ulazni signal `iSIGN` ima vrednost 0. U tom slučaju upravljačka jedinica nakon aktiviranja ulaznog signala za sabiranje, `iQA=1`, prelazi iz stanja `sT0` u stanje `sT2`. U tom stanju se analizira vrednost ulaznog signala `iSIGN` i pošto je on na niskom nivou upravljačka jedinica prelazi u stanje `sT3` gde se izvršava sabiranje. Nakon toga UJ se vraća u početno stanje `sT0` gde čeka sledeću komandu za sabiranje ili oduzimanje brojeva.

Sledeća prikazana situacija odgovara slučaju kada je `iSIGN` jednak jedinici (operandi imaju različite znakove) i ulazni signal `iE` ima vrednost nula. U tom slučaju UJ izvršava sledeću sekvencu stanja: `sT0`, `sT2`, `sT4`, `sT5`, `sT6`, `sT7` i ponovo `sT0`. Ovo je najduža putanja grafa prelaza stanja, Slika 8.56, prilikom izvršenja operacije sabiranja.

Poslednja moguća situacija prilikom sabiranja je kada su ulazni signali `iSIGN` i `iE` na viskom nivou. Tada se izvršava sledeća sekvenca prelazaka stanja: `sT0`, `sT2`, `sT4`, `sT5` i ponovo `sT0`.

U sva tri slučaja vrednosti izlaznih signala u svakom stanju su usklađeni sa vrednostima koje prikazuje Tabela 8.17.

Vremenski dijagram izvršenja operacija oduzimanja brojeva prikazuje Slika 8.58. Prikazane su tri moguće situacije u zavisnosti od vrednosti ulaznih signala `iSIGN` i `iE`. Prvo je prikazana situacija `iE=iSIGN=1`, nakon čega sledi situacija kada je `iE` na niskom nivou dok `iSIGN` ostaje na viskom nivou. Poslednja situacija je kada je signal `iSIGN` na niskom nivou. U tom slučaju vrednost ulaznog signala `iE` nije bitna (na dijagramu je `iE=0`). Prelasci stanja su slični kao i kod sabiranja, sa tom razlikom da je u ovom slučaju prvo stanje, nakon početnog stanja `sT0`, uvek `sT1`. Iz ovog stanja se prelazi u stanje `sT2` i dalje je situacija identična kao kod sabiranja brojeva.



Slika 8.58: Vremenski dijagram oduzimanja brojeva

Na osnovu grafa prelazaka stanja upravljačke jedinice, Slika 8.56, moguće je formirati automat koji izvršava zadati upravljački algoritam. U tom slučaju upravljačka jedinica je zapravo Murov automat gde vrednost izlaznih signala zavisi isključivo od trenutnog stanja, dok prelasci stanja zavise od vrednosti ulaznih signala i trenutnog stanja upravljačke jedinice.

Realizacija arhitekture automata i u ovom slučaju ima tri dela:

1. kombinaciona mreža za određivanje narednog stanja,
2. flip-flopovi za memorisanje trenutnog stanja (stanja su u ovom slučaju kodovana binarno) i
3. dekoderska logika za određivanje vrednosti izlaznih, upravljačkih signala.

Može se primetiti da zbog potrebe za dekodovanje trenutnog stanja obe kombinacione mreže će biti kompleksnije nego što je to slučaj kod realizacije upravljačke jedinice metodom 1 memorijki element po stanju gde je dekodovanje trenutnog stanja znatno jednostavnije.

U nastavku je prikazan VHDL kod implementacije upravljačke jedinice za sabiranje označenih brojeva realizovan na osnovu grafa prelaza stanja koji prikazuje Slika 8.56.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY UJ2 IS PORT(
  iCLK, inRESET,
  iE, iSIGN,
  iQA, iQS:       IN  STD_LOGIC;
  oX, oC_IN, oL,
  oY, oZ, oW:     OUT STD_LOGIC;
  oALJ_SEL:       OUT STD_LOGIC_VECTOR(2 DOWNTO 0) );
END UJ2;

ARCHITECTURE ARH_UJ2 of UJ2 IS
  -- deklaracija tipa sa svim stanjima koje ima UJ
  TYPE   tUJ_STATE is (T0, T1, T2, T3, T4, T5, T6, T7);

  -- deklaracija signala koji sadrze informaciju o
  -- tekucem i sledecem (narednom) stanju
  SIGNAL sSTANJE, sSLEDECE_STANJE: tUJ_STATE;
  -- izlazni vektor od 9 signala
  SIGNAL sIZLAZ: STD_LOGIC_VECTOR(8 DOWNTO 0);
BEGIN
  -- kombinaciona mreza koja na osnovu trenutnog stanja
  -- i vrednosti ulaza generise kod narednog stanja
  PROCESS (sSTANJE, iQA, iQS, iSIGN, iE) BEGIN
    CASE sSTANJE IS
      WHEN T0 => IF (iQS = '1') THEN
                   sSLEDECE_STANJE <= T1;
                 ELSIF (iQA = '1') THEN
                   sSLEDECE_STANJE <= T2;
                 ELSE
                   sSLEDECE_STANJE <= T0;
```

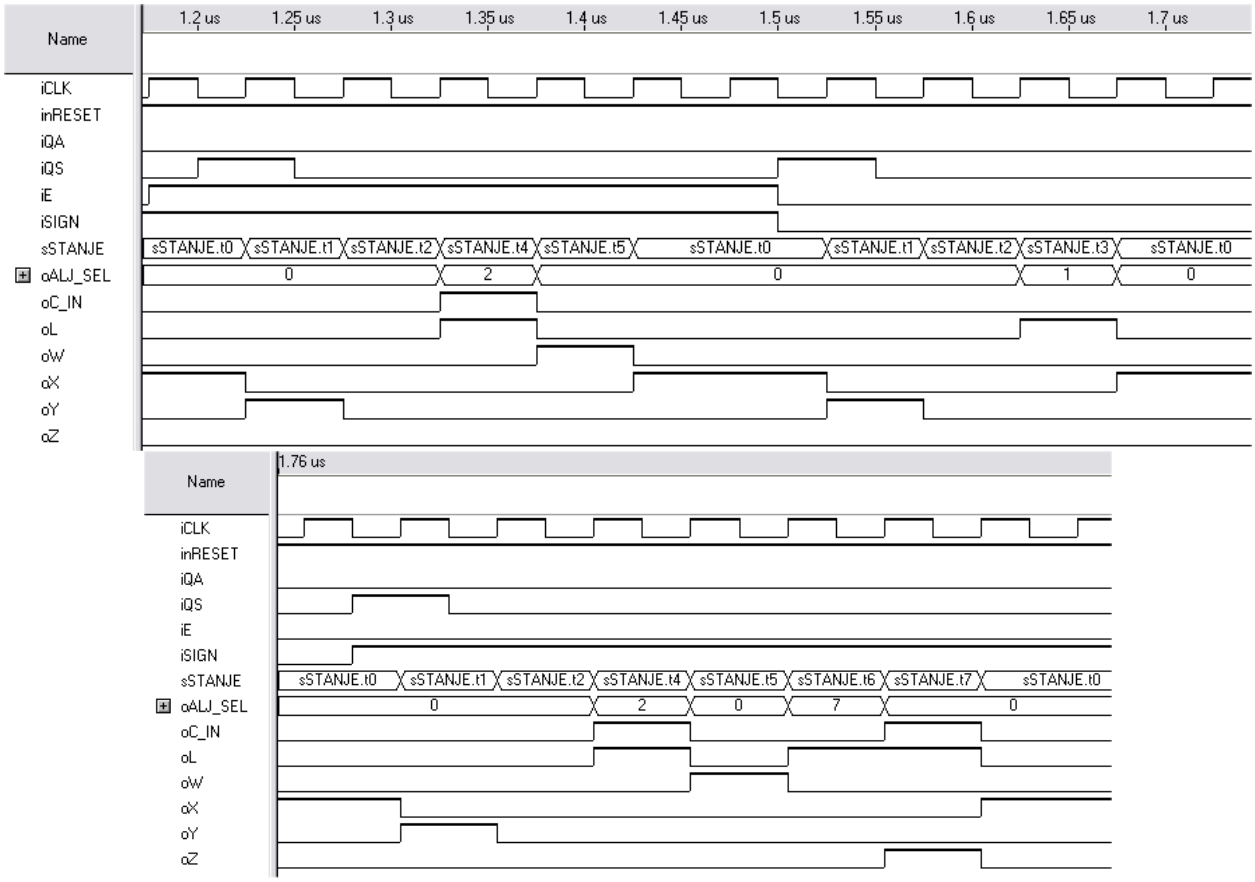
```
                END IF;
    WHEN T1 => sSLEDECE_STANJE <= T2;
    WHEN T2 => IF (iSIGN = '1') THEN
                 sSLEDECE_STANJE <= T4;
               ELSE
                 sSLEDECE_STANJE <= T3;
               END if;
    WHEN T3 => sSLEDECE_STANJE <= T0;
    WHEN T4 => sSLEDECE_STANJE <= T5;
    WHEN T5 => IF (iE = '1') THEN
                 sSLEDECE_STANJE <= T0;
               ELSE
                 sSLEDECE_STANJE <= T6;
               END IF;
    WHEN T6 => sSLEDECE_STANJE<= T7;
    WHEN T7 => sSLEDECE_STANJE <= T0;
  END CASE;
END PROCESS;

-- flip-flopovi za smestanje koda
-- trenutnog stanja; postavljanje pocetnog
-- stanja je sinhrono sa signalom takta
PROCESS (iCLK) BEGIN
  IF (iCLK'event and iCLK='1') THEN
    IF (inRESET = '0') then  -- sinhroni reset
      sSTANJE <= T0;
    ELSE
      sSTANJE <= sSLEDECE_STANJE;
    END IF;
  END IF;
END PROCESS;

-- formiranje izlaznog vektora na osnovu tekuceg stanja
PROCESS (sSTANJE) BEGIN
   CASE sSTANJE IS
     WHEN T0 => sIZLAZ <= "100000000";
     WHEN T1 => sIZLAZ <= "000000100";
     WHEN T2 => sIZLAZ <= "000000000";
     WHEN T3 => sIZLAZ <= "000101000";
     WHEN T4 => sIZLAZ <= "001011000";
     WHEN T5 => sIZLAZ <= "000000001";
     WHEN T6 => sIZLAZ <= "011101000";
     WHEN T7 => sIZLAZ <= "000011010";
   END CASE;
 END PROCESS;

 -- odredjivanje vrednosti izlaznih signala
 -- na osnovu vrednosti izlaznog vektora
 oX       <= sIZLAZ(8);
 oALJ_SEL <= sIZLAZ(7 DOWNTO 5);
 oC_IN    <= sIZLAZ(4);
 oL       <= sIZLAZ(3);
 oY       <= sIZLAZ(2);
 oZ       <= sIZLAZ(1);
 oW       <= sIZLAZ(0);

END ARH_UJ2;
```

Slika 8.59 i Slika 8.60 redom prikazuju vremenske dijagrame izvršenja operacija sabiranja i oduzimanja brojeva. Na vremenskim dijagramima su prikazani svi mogući slučajevi prelazaka stanja. Isti slučajevi su analizirani u implementaciji upravljačke jedinice metodom 1 memorijski element po stanju.

Slika 8.59: Vremenski dijagram sabiranja brojeva

Sumirajući rezultate realizacije tražene upravljačke jedinice na zadata dva načina može se videti da realizacija UJ metodom 1 memorijski element po stanju rezerviše ukupno 14 logičkih elemenata (eng. LE – *Logic Element*) prigramibilne mreže ALTERA FLEX 10K30. Ista upravljačka jedinica realizovana pomoću automata u istoj programibilnoj mreži zauzima 44 logička elementa. Ovakvo znatno uvećanje zauzetih resursa programinilne mreže je bilo očekivano zbog znatno složenije dekoderske logike za određivanje trenutnog stanja u slučaju realizacije pomoću automata.

Maksimalna radna frekvencija je u prvom slučaju ograničena na 250MHz zbog fizičkih karakteristika same programibilne mreže, dok je kod realizacije UJ automatom maksimalna radna frekvencija 112,36MHz. Zbog složenijih kombinacionih mreža za pobudu flip-flopova kod realizacije UJ automatom putanje signala od izlaza flip-flopa do njegovog ulaza su znatno duže. Zbog toga je veće vreme propagacije signala kroz kombinacionu mrežu, što traži dužu periodu takt signala radi stabilizacije signala pobude flip-flopova. Zbog toga je maksimalna radna frekvencija u ovom slučaju primetno manja u odnosu na realizaciju UJ metodom 1 memorijski element po stanju.



Slika 8.60: Vremenski dijagram oduzimanja brojeva

Slika 8.61 prikazuje poređenje vremenskih parametara kašnjenja izlaznih signala u odnosu na takt signal za obe realizacije upravljačke jednice. Zbog jednostavnijeg dekodovanja izlaznih signala kašnjenje izlaznih signala u odnosu na takt signal je u proseku manje za oko 5ns kod realizacije UJ metodom 1 memorijski element po stanju.

tco (Clock to Output Delays)

Output Name -- Register Name -- Clock Name	Actual tco
oALJ_SEL[0]	8.300 ns
oALJ_SEL[1]	7.600 ns
oALJ_SEL[2]	6.200 ns
oC_IN	9.000 ns
oL	9.100 ns
oW	6.200 ns
oX	7.400 ns
oY	7.400 ns
oZ	6.200 ns

a)

tco (Clock to Output Delays)

Output Name -- Register Name -- Clock Name	Actual tco
oL	14.300 ns
oALJ_SEL[0]	14.100 ns
oALJ_SEL[2]	13.900 ns
oW	13.800 ns
oY	13.200 ns
oZ	13.200 ns
oX	12.700 ns
oC_IN	12.700 ns
oALJ_SEL[1]	12.400 ns

b)

Slika 8.61: Kašnjenje izlaznih signala u odnosu na takt signal
a) kod realizacije UJ metodom 1 memorijski element po stanju
b) kod realizacije UJ pomoću automata

8.14 ZADATAK:

Projektovati mikroprogramsku UJ za sabiranje označenih brojeva prema zadatom algoritmu (Slika 8.53). Isprojektovanu upravljačku jedinicu realizovati u VHDL jeziku za opis fizičke arhitekture.

REŠENJE:

Sa ciljem projektovanja mikroprogramske upravljačke jedinice za sabiranje označenih brojeva, prvo je potrebno na osnovu blok dijagrama algoritma (Slika 8.53) napisati mikroprogram u RTLu. Mikroprogram se interpretira kao redosled izvršavanja mikrooperacija.

Adresa	Mikroinstrukcija
0	$X = 1; \text{if}\,(q_S = 1)\,\text{then}\,(\text{goto1}),\ \text{if}\,(q_a = 1)\,\text{then}\,(\text{goto2})$ $\text{if}\,((q_S \vee q_a) = 0)\,\text{then}\,(\text{goto0})$
1	$X = 0;\ B_S \leftarrow \overline{B_S}$
2	$X = 0;\ \text{if}\,(S = 1)\,\text{then}\,(\text{goto4})$
3	$X = 0;\ A \leftarrow A + B, E \leftarrow C_{izl}, \text{goto0}$
4	$X = 0;\ A \leftarrow A + \overline{B} + 1, E \leftarrow C_{izl}$
5	$X = 0;\ \text{if}\,(E = 1)\,\text{then}\,(\text{goto0}), E \leftarrow 0$
6	$X = 0;\ A \leftarrow \overline{A}$
7	$X = 0;\ A \leftarrow A + 1, A_S \leftarrow \overline{A_S}, (\text{goto } 0)$

Na osnovu niza mikrooperacija može se uočiti da mikroprogramska upravljačka jedinica iz ovog zadatka ima 8 stanja, tri adresne linije i četiri spoljna ulaza (q_a, q_s, S i E). Njen format instrukcije je sledeći:

X	S_2	S_1	S_0	C_{in}	L	y	z	w	ADRESA			G_1	G_0
1	2	3	4	5	6	7	8	9	10	11	12	13	14

Signali G_1 i G_0 predstavljaju upravljačke signale na osnovu kojih generator sledeće adrese upravlja adresnim registrom. Na osnovu ovih signala generator sledeće adrese vrši test odluke koji prikazuje Tabela 8.18.

G_1	G_0	Odluka
0	0	uvećanje sadržaja adresnog registra
0	1	paralelni unos u adresni registar
1	0	unos u adresni registar za S=1 a uvećanje sadržaja adresnog registra za S=0
1	1	unos u adresni registar za E=1 a uvećanje sadržaja adresnog registra za E=0

Tabela 8.18: Test odluke za upravljanje adresnim registrom

Pošto je broj stanja jednak 8, potreban je brojač sa paralelnim upisom koji sadrži tri memorijska elementa. Stanja automata odgovaraju adresama, a na adresama mikroprogramske memorije su smeštane mikroinstrukcije, ili kontrolne reči prema uvedenom formatu. Stalna memorija u koju se smeštaju mikroinstrukcije naziva se i upravljačkom memorijom. Njen sadržaj za primer sabiranja označenih brojeva je dat u obliku programske tablice ROMa (Tabela 8.19).

Adresa ROMa			X	S_2	S_1	S_0	C_{in}	L	y	z	w	A_2	A_1	A_0	G_1	G_0
			1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0	0	0	1	0	0	1
0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0
0	1	1	0	0	0	1	0	1	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	1	1	0	0	0	1	0	1	0	1
1	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	1
1	1	0	0	1	1	1	0	1	0	0	0	1	1	1	0	1
1	1	1	0	0	0	0	1	1	0	1	0	0	0	0	0	1

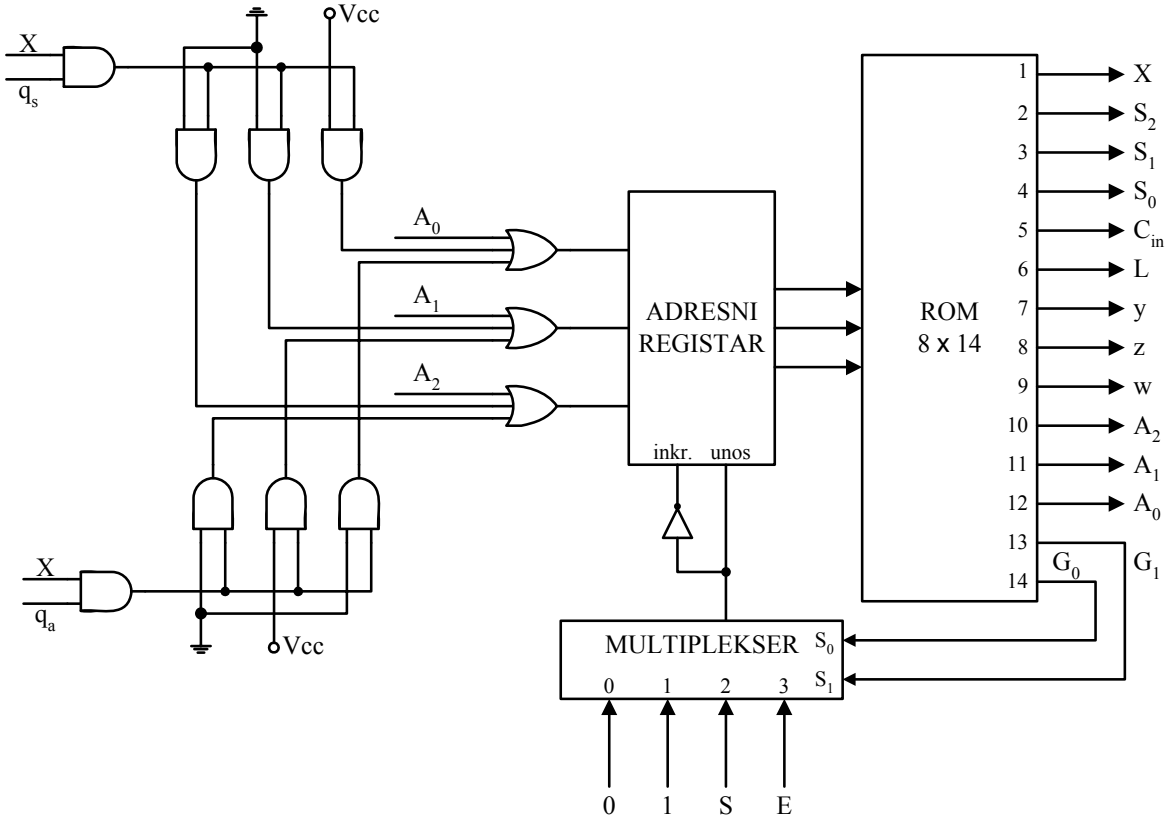
Tabela 8.19: Programska tablica mikroprogramskog ROMa

Na adresi 000 selekcione promenljive su 01 da bi se omogućio unos spoljne adrese u adresni registar (001 ili 010). Ukoliko nema spoljne adrese sledeća adresa je 000. Na adresi 001 vrši se komplement znaka operanda B, pa je zato Y=1, selekcione promenljive 01 da se omogući unos, a sledeća adresa 010. Prateći dalje mikroprogram, uočava se da se na adresi 010 ispituje da li je S=1. Zbog toga su selekcione promenljive 10, a sledeća adresa 100 (gde se vrši oduzimanje). Na adresi 011, kod operacije $S_2S_1S_0$ je 001, jer se vrši sabiranje operanada A i B, L=1, jer se vrši upis u akumulator, selekciona promenljiva 01 zbog učitavanja sledeće adrese, a sledeća adresa 000. Na adresi 100 vrši se oduzimanje, pa je kod operacije 010, C_{in}=1 (zbog $A - B = A + \overline{B} + 1$), L=1 jer se vrši upis u akumulator, sledeća adresa 101, a selekciona promenljiva 01. Na adresi 101 vrši se provera uslova za vrednost E, pa je zato selekciona promenljiva 11, W=1. Na adresi 110, vrši se invertovanje sadržaja akumulatora, pa je kod operacije 111, a selekcija 01. I na kraju, na adresi 111, kod operacije je 000, C_{in}=1 jer imamo operaciju uvećanja A za 1, Z=1 zbog invertovanja znaka A_S, A sledeća adresa 000.

Slika 8.62 prikazuje blok šemu upravljačke jedinice za sabiranje označenih brojeva.

Ulaz definiše spoljnu adresu. Uzima se da je u slučaju pojave spoljnog signala q_a =1 adresa mikroinstrukcije 010, u slučaju q_s =1 adresa 001. Adresa 000 se odnosi na slučaj kada nema spoljnog signala i kad je operacija završena, tj X=1. Sadržaj adresnog registra definiše adresu mikroinstrukcije. Selekcione promenljive, biti 13 i 14, vode se na multiplekser koji izabira jednu od četiri

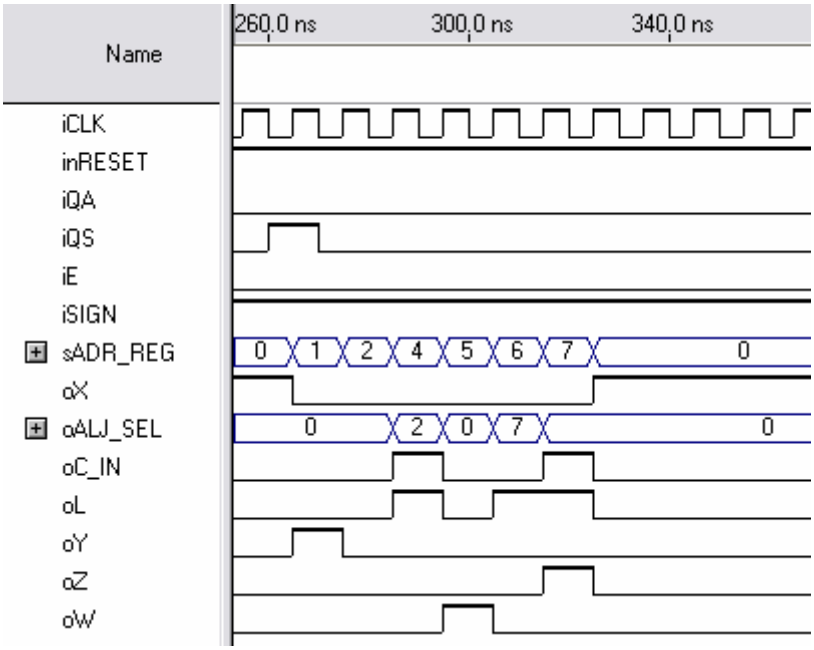
moguće mikrooperacije. U opštem slučaju za selekcione promenljive sa vrednošću 01, adresno polje je sledeća adresa. U slučaju da su one 10 selektuje se promenljiva S, a za slučaj 11 promenljiva E. Za ova dva slučaja sledeća adresa je ona koja je definisana za slučaj da je vrednost (S, E) bila jednaka 1. U slučaju da taj bit nije postavljen, sledeća adresa je naredna adresa u nizu, jer se sadržaj adresnog registra uvećava za 1. Vidimo da je uvećanje sadržaja adresnog registra ekvivalentno izboru sledeće po redu mikroinstrukcije.



Slika 8.62: Blok šema upravljačke jedinice za sabiranje označenih brojeva

Na osnovu prikazane blok šeme može se realizovati VHDL opis upravljačke jedinice. Kao što prikazuje Slika 8.62 VHDL opis se sastoji iz tri celine: multiplekser za kontrolu adresnog registra, adresni registar i mikroprogramska memorija. Ulazi u digitalni sistem direktno utiču na izlaznu funkciju adresnog registra, dok se izlazne mikrooperacije direktno izdvajaju iz adresirane mikroinstrukcije.

Prema istim principima kao i u prethodnom zadatku može se izvršiti simulacija rada isprojektovane upravljačke jedinice. Trenutno stanje upravljačke jedinice je izraženo stanjem adresnog registra. Adresirana mikroinstrukcija određuje vrednost izlaznih signala definišući sve potrebne mikrooperacije za izvršenje trenutne mikroinstrukcije. Tako na primer Slika 8.63 prikazuje izvršenje mikroalgoritma oduzimanja za ulaze E=0 i S=1.



Slika 8.63: Primer izvršenja mikroalgoritma oduzimanja

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.std_logic_arith.all;

ENTITY uPROG_UJ1 IS
  PORT (
  iCLK, inRESET,
  iE, iSIGN,
  iQA, iQS:       IN  STD_LOGIC;
  oX, oC_IN, oL,
  oY, oZ, oW:     OUT STD_LOGIC;
  oALJ_SEL:       OUT STD_LOGIC_VECTOR(2 DOWNTO 0) );
END uPROG_UJ1;

ARCHITECTURE ARH_uPROG_UJ OF uPROG_UJ1 IS
  -- deklaracija mikroprogramske memorije
  SIGNAL sINSTRUCTION: STD_LOGIC_VECTOR(13 DOWNTO 0);
  -- adresni registar
  SIGNAL sADR_REG: UNSIGNED(2 DOWNTO 0);
  -- dozvola paralelnog upisa u adr. reg.
  SIGNAL sLOAD    : STD_LOGIC;
  -- dozvola uvecanja adresnog registra
  SIGNAL sINCR    : STD_LOGIC;

  -- interne mikrooperacije
  ----------------------------------------
  -- indikacija izvrsenja mikroalgoritma
  SIGNAL sX       : STD_LOGIC;
  -- adresa naredne mikroinstrukcije
  SIGNAL sADR     : STD_LOGIC_VECTOR(2 DOWNTO 0);
  -- adresiranje multipleksera za kotrolu adresnog registra
  SIGNAL sG       : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
```

```
-- multiplekser za kontrolu adresnog registra
--   izlaz je signal za dozvolu paralelnog upisa
--   adrese sledece mikroinstrukcije
PROCESS (sG, iE, iSIGN) BEGIN
  CASE sG IS
    WHEN "00"   => sLOAD <= '0';
    WHEN "01"   => sLOAD <= '1';
    WHEN "10"   => sLOAD <= iSIGN;
    WHEN "11"   => sLOAD <= iE;
    WHEN OTHERS => sLOAD <= '0';
  END CASE;
END PROCESS;

-- signal dozvole uvecanja adresnog registra za 1
sINCR <= NOT sLOAD;

-- adresni registar
PROCESS (iCLK, inRESET) BEGIN
  IF (inRESET = '0') THEN
    sADR_REG <= "000"; -- postavljanje pocetnog stanja, asinhrono
  ELSIF (iCLK'EVENT AND iCLK = '1') THEN
    -- pocetak izvrsenja mikroalgoritma
    IF (sX = '1') THEN
      -- paralelni upis
      IF (sLOAD = '1') THEN
        IF (iQA = '1') THEN    -- sabiranje
          sADR_REG <= "010";
        ELSIF (iQS = '1') THEN -- oduzimanje
          sADR_REG <= "001";
        END IF;
      END IF;
    ELSE
    -- izvrsenje mikroalgoritma je u toku
      -- paralelni upis
      IF (sLOAD = '1') THEN
        sADR_REG <= UNSIGNED(sADR);
      END IF;

      -- uvecanje adrese za jedan
      IF (sINCR = '1') THEN
        sADR_REG <= sADR_REG + 1;
      END IF;
    END IF;
  END IF;
END PROCESS;

-- mikroprogramska memorija
PROCESS (sADR_REG) BEGIN
  CASE sADR_REG IS
    WHEN  "000" => sINSTRUCTION <= "10000000000001";
    WHEN  "001" => sINSTRUCTION <= "00000010001001";
    WHEN  "010" => sINSTRUCTION <= "00000000010010";
    WHEN  "011" => sINSTRUCTION <= "00010100000001";
    WHEN  "100" => sINSTRUCTION <= "00101100010101";
    WHEN  "101" => sINSTRUCTION <= "00000000100011";
    WHEN  "110" => sINSTRUCTION <= "01110100011101";
    WHEN  "111" => sINSTRUCTION <= "00001101000001";
    WHEN OTHERS => sINSTRUCTION <= "10000000000001";
```

```
    END CASE;
  END PROCESS;

  -- izdvajanje mikrooperacija iz adresirane mikroinstrukcije
  sG       <= sINSTRUCTION(1 DOWNTO 0);
  sADR     <= sINSTRUCTION(4 DOWNTO 2);
  oW       <= sINSTRUCTION(5);
  oZ       <= sINSTRUCTION(6);
  oY       <= sINSTRUCTION(7);
  oL       <= sINSTRUCTION(8);
  oC_IN    <= sINSTRUCTION(9);
  oALJ_SEL <= sINSTRUCTION(12 DOWNTO 10);
  oX       <= sINSTRUCTION(13);

  -- interni signal za kontrolu adresnog registra
  sX       <= sINSTRUCTION(13);

END ARH_uPROG_UJ;
```

8.15 ZADATAK:

Modifikovati mikroprogramsku jedinicu iz prethodnog zadatka tako da vrši sabiranje/oduzimanje pozitivnih i negativnih brojeva, pri čemu su negativni brojevi predstavljeni dvostrukim komplementom. Uključiti detekciju prekoračenja.

Slika 8.64 prikazuje blok dijagram algoritma za sabiranje označenih brojeva predstavljenim dvostrukim komplementom.

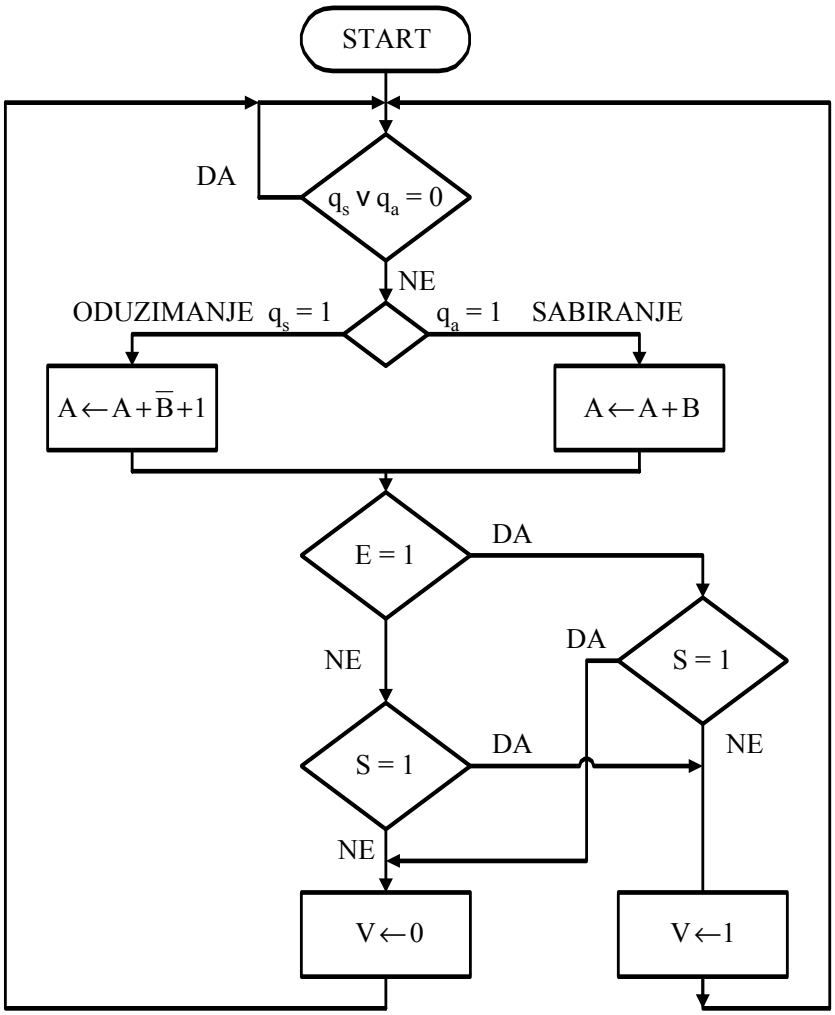
Za implementaciju mikroprogramske memorije u VHDL opisu upravljačke jedinice iskoristiti ALTERA megafunkciju `LPM_ROM`.

REŠENJE:

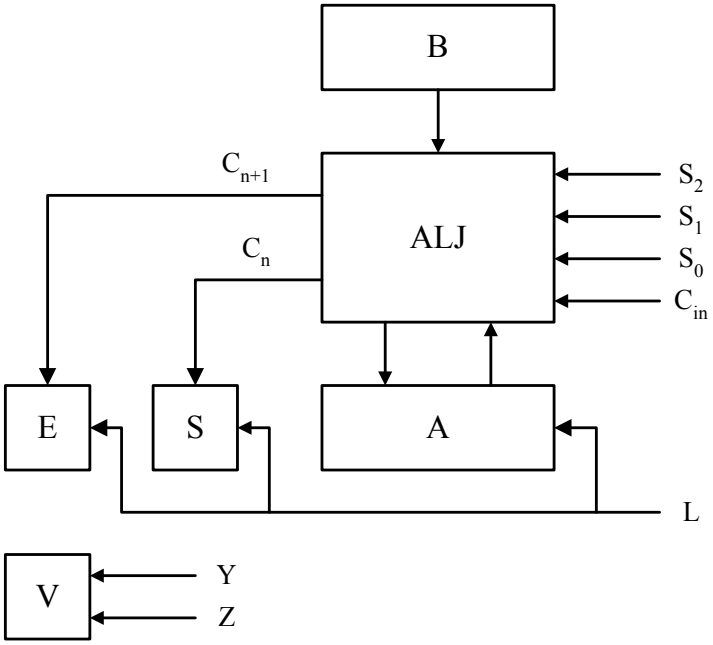
Slika 8.65 prikazuje mikroprocesor sa komponentama potrebnim za realizaciju traženog algoritma. Takođe, prikazani su i signali koje generiše upravljačka jedinica. Slika 8.66 prikazuje ulazno-izlazne signale upravljačke jedinice za sabiranje pozitivnih i negativnih brojeva prema algoritmu koji prikazuje Slika 8.64.

Kod određivanja prekoračenja bitna je vrednost izlaznog prenosa, C_{n+1}, i vrednost bita prenosa između dva najviše značajna bita, C_n. Na blok dijagramu procesora ova dva bita su predstavljenja statusnim registrima E i S respektivno. Stanje ova dva statusna registra se osvežava zajedno sa upisom trenutne vrednosti rezultata u akumulatorski registar A. Izvršenje mikroinstrukcije upisa u ova tri registra se definiše uz pomoć upravljačkog signala L.

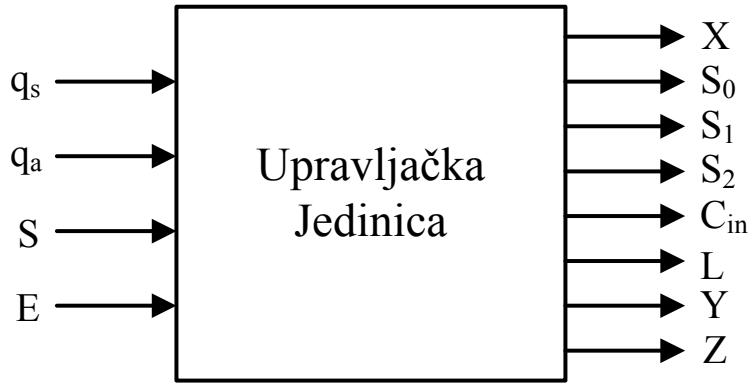
Vrednost bita prekoračenja smešta se u statusni registar sa oznakom V. Upravljački signali Y i Z služe za postavljanje bita prekoračenja na vrednost 1 (Y=1 => V=1), odnosno na vrednost 0 (Z=1 => V=0).



Slika 8.64: Blok dijagram algoritma za sabiranje označenih brojeva sa uključenom detekcijom prekoračenja



Slika 8.65: Blok dijagram mikroprocesora



Slika 8.66: Ulazno-izlazni signali upravljačke jedinice

Sa ciljem projektovanja mikroprogramske upravljačke jedinice za sabiranje označenih brojeva predstavljenim dvostrukim komplementom, na osnovu blok dijagram algoritma (Slika 8.64) može se napisati mikroprogram u RTLu. Tabela 8.20 prikazuje upravljački mikroprogram koji se interpretira kao redosled izvršavanja mikrooperacija.

Adresa	Mikroinstrukcija
0	X=1; IF (q_s=1) GOTO 1, IF (q_a=1) GOTO 2
1	X=0; A←A+not(B)+1, S←C_n, E←C_{n+1}, GOTO 3
2	X=0; A←A+B, S←C_n, E←C_{n+1}
3	X=0; IF (E=1) GOTO 6
4	X=0; IF (S=1) GOTO 7
5	X=0; V←0 : GOTO 0
6	X=0; IF (S=1) GOTO 5
7	X=0; V←1 : GOTO 0

Tabela 8.20: Mikroprogram

Na osnovu niza mikrooperacija primećuje se da mikroprogramska upravljačka jedinica iz ovog zadatka ima 8 stanja, tri adresne linije i četiri spoljna ulaza (q_a, q_s, S i E). Njen format instrukcije je sledeći:

X	S_2	S_1	S_0	C_{ul}	L	Y	Z	A_2	A_1	A_0	G_1	G_0
1	2	3	4	5	6	7	8	9	10	11	12	13

Signali G_1 i G_0 predstavljaju upravljačke signale na osnovu kojih generator sledeće adrese upravlja adresnim registrom. Na osnovu ovih signala generator sledeće adrese vrši test odluke koji prikazuje Tabela 8.21.

G_1	G_0	Odluka
0	0	RM ←RM+1 (inc)
0	1	RM←ULAZ (load)
1	0	IF (S=1) THEN (RM←ULAZ) ELSE (RM←RM+1)
1	1	IF (E=1) THEN (RM←ULAZ) ELSE (RM←RM+1)

Tabela 8.21: Test odluke za upravljanje adresnim registrom

Pošto je broj stanja jednak 8, potreban je brojač sa paralelnim upisom koji se sastoji od tri memorijska elementa. Stanja automata odgovaraju adresama, a na adresama ROMa su smeštane mikroinstrukcije, ili kontrolne reči prema uvedenom formatu. Stalna memorija u koju se smeštaju mikroinstrukcije naziva se i upravljačka memorija. Njen sadržaj za primer sabiranja označenih brojeva je dat u obliku programske tablice ROMa (Tabela 8.22).

Adresa	X	S_2	S_1	S_0	C_{in}	L	Y	Z	A_2	A_1	A_0	G_1	G_0
0	1	×	×	×	×	0	0	0	0	0	0	0	1
1	0	0	1	0	1	1	0	0	0	1	1	0	1
2	0	0	0	1	0	1	0	0	0 ×	1 ×	1 ×	0 0	1 0
3	0	×	×	×	×	0	0	0	1	1	0	1	1
4	0	×	×	×	×	0	0	0	1	1	1	1	0
5	0	×	×	×	×	0	0	1	0	0	0	0	1
6	0	×	×	×	×	0	0	0	1	0	1	1	0
7	0	×	×	×	×	0	1	0	0	0	0	0	1

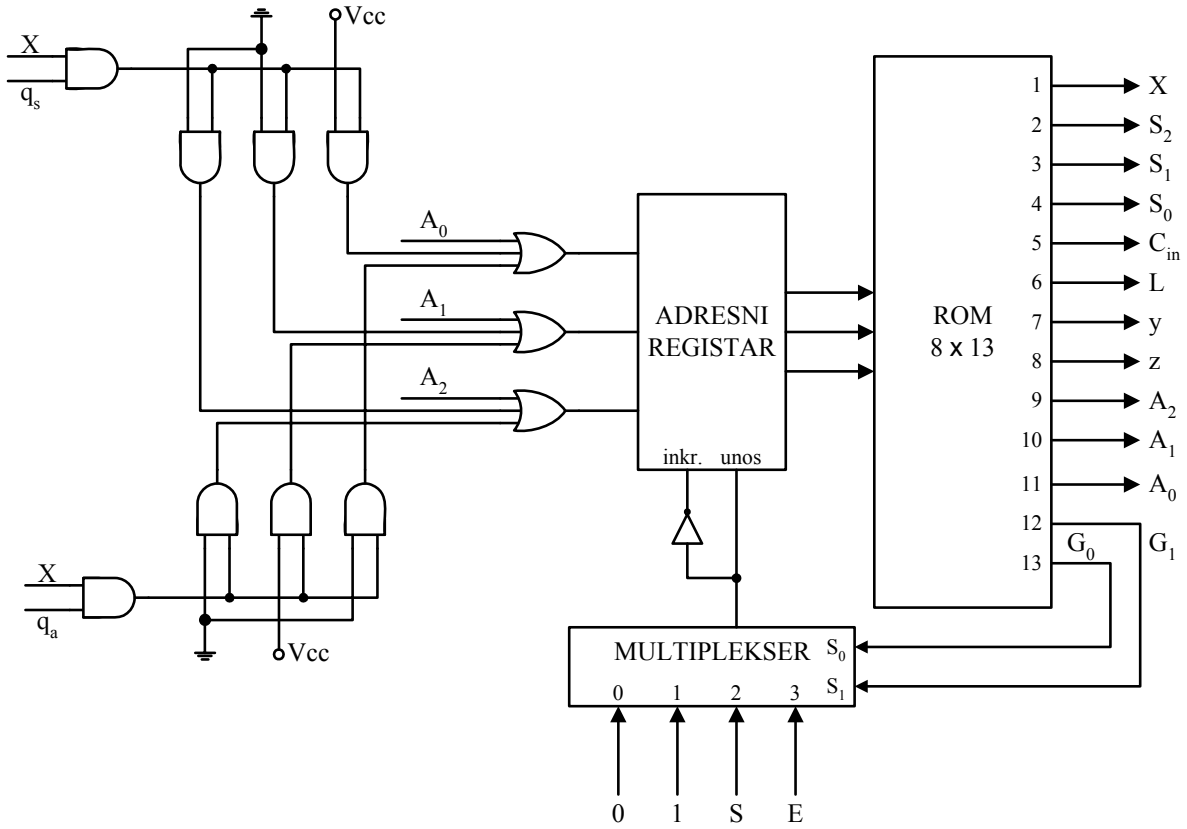
Tabela 8.22: Programska tablica

U slučaju da je izabrana operacija oduzimanja (q_s=1), tada se izvršava sekvenca na adresi 1. Ako su nakon izvršene operacije i E i S jednaki (oba su ili "0" ili "1"), tada nema prekoračenja, i V dobija vrednost 0. Ako su E i S različiti, V je "1". Slično važi i za operaciju sabiranja.

Slika 8.67 prikazuje blok šemu realizovane mikroprogramske upravljačke jedinice.

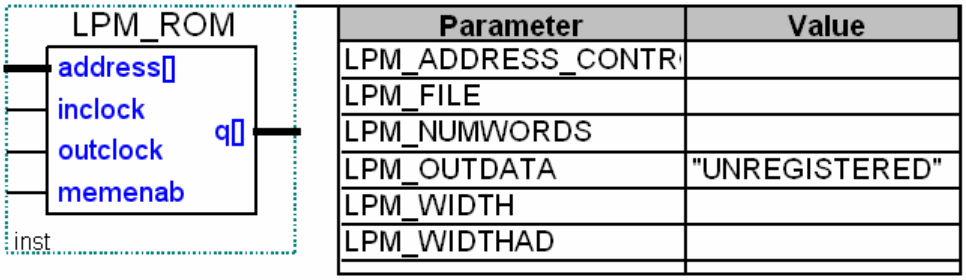
Prilikom realizacije generatora sledeće adrese uzima se da je u slučaju pojave spoljnog signala q_s=1 adresa naredne mikroinstrukcije 1, a u slučaju q_a=1 naredna mikroinstrukcija se nalazi na adresi 2. Na adresi 0, selekcione promenljive G_1 i G_0 su 01, kako bi se omogućio unos spoljne adrese u adresni registar (001 ili 010). Ukoliko nema spoljne adrese, sledeća adresa je 000. Operacija bezuslovnog skoka GOTO se realizuje tako što se selekcione promenljive G_1 i G_0 postave na 01, čime se aktivira signal paralelnog unosa u adresni registar, a zatim se učitava adresa na kojoj se nastavlja izvršavanje programa (koja je prisutna na signalima A_2, A_1 i A_0).

Ukoliko nema bezuslovnog skoka, već se naredna mikrooperacija nastavlja na sledećoj adresi, tada postoje dva izbora. Jedan je da pomoću signala za paralelni unos u adresni registar učitamo narednu adresu. Drugi način je da se selekcione promenljive G_1 i G_0 postave na 00, čime se aktivira signal za uvećavanje sadržaja adresnog registra za 1.



Slika 8.67: Struktura mikroprogramske upravljačke jedinice

Na osnovu strukturnog prikaza isprojektovane mikroprogramske upravljačke jedinice može se realizovati VHDL opis iste. Za realizaciju mikroprogramske memorije prema tekstu zadatka treba iskoristi ALTERA megafunkciju `LPM_ROM`. Uz pomoć ove funkcije, mikroprogramska memorija će se direktno implementirati u namenske matrice memorijskih elemenata koje postoje u ALTERA programabilnim sekvencijalnim mrežama.



Parameter	Value
LPM_ADDRESS_CONTR	
LPM_FILE	
LPM_NUMWORDS	
LPM_OUTDATA	"UNREGISTERED"
LPM_WIDTH	
LPM_WIDTHAD	

Slika 8.68: Blok dijagram `LPM_ROM` megafunkcije

Slika 8.68 prikazuje blok dijagram ove komponente, gde se vidi lista mogućih ulaznih i izlaznih portova kao i grupa karakterističnih parametara. Na izlaznoj magistrali podataka `q` se nalazi vrednost trenutno adresirane lokacije preko ulazne adresne magistrale `address`. Adresna magistrala može biti interno registrovana sa taktom `inclock`, dok se za registrovanje izlazne magistrale podataka može iskoristiti takt `outclock`. Ulazni port `memenab` služi za dozvolu izlaza podataka iz memorije, `memenab=1`, dok je u slučaju kada je `memenab=0` izlazna magistrala podataka u stanju visoke impedance. Od svih ulaznih i izlaznih portova obavezni su samo `address` i `q`. Ostali se mogu iskoristi prema potrebi.

Arhitektura instancirane memorije se konfiguriše pomoću parametara. Tako na primer parametrom `LPM_WIDTH` se definiše širina magistrale podataka, dok se parametrom `LPM_WIDTHAD` određuje širina adresne magistrale. Parametar `LPM_FILE` se koristi za ukazivanje na datoteku koja definiše sadržaj ROM memorije. Vrednost ova tri parametra se mora definisati prilikom instanciranja memorije, dok su ostali opcioni. Detaljan opis se može pronaći u dokumentaciji koja se isporučuje sa *Altera Quartus II* programskim paketom.

Celokupna VHDL deklaracija ove komponente sa inicijalnim vrednostima generičkih parametara je sledeća:

```
COMPONENT lpm_rom
  GENERIC (
   LPM_WIDTH              : POSITIVE;
   LPM_WIDTHAD            : POSITIVE;
   LPM_NUMWORDS           : NATURAL   := 0;
   LPM_ADDRESS_CONTROL    : STRING    := "REGISTERED";
   LPM_OUTDATA            : STRING    := "REGISTERED";
   LPM_FILE               : STRING;
   LPM_TYPE               : STRING    := "LPM_ROM";
   LPM_HINT               : STRING    := "UNUSED"
  );
  PORT (
   address            : IN STD_LOGIC_VECTOR(LPM_WIDTHAD-1 DOWNTO 0);
   inclock, outclock  : IN STD_LOGIC := '0';
   memenab            : IN STD_LOGIC := '1';
   q                  : OUT STD_LOGIC_VECTOR(LPM_WIDTH-1 DOWNTO 0)
  );
END COMPONENT;
```

VHDL opis mikroprogramske memorije je veoma sličan rešenju iz prethodnog zadatka. Razlika je samo u načinu realizacije mikroprogramske memorije.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.std_logic_arith.all;
LIBRARY lpm;
  USE lpm.lpm_components.all;
```

```
ENTITY uPROG_UJ2 IS
  PORT (
  iCLK, inRESET,
  iE, iSIGN,
  iQA, iQS:      IN  STD_LOGIC;
  oX, oC_IN,
  oL, oY, oZ:    OUT STD_LOGIC;
  oALJ_SEL:      OUT STD_LOGIC_VECTOR(2 DOWNTO 0) );
END uPROG_UJ2;

ARCHITECTURE ARH_uPROG_UJ OF uPROG_UJ2 IS
  -- deklaracija mikroprogramske memorije
  SIGNAL sINSTRUCTION: STD_LOGIC_VECTOR(12 DOWNTO 0);
  -- adresni registar
  SIGNAL sADR_REG: UNSIGNED(2 DOWNTO 0);
  -- dozvola paralelnog upisa u adr. reg.
  SIGNAL sLOAD   : STD_LOGIC;
  -- dozvola uvecanja adresnog registra
  SIGNAL sINCR   : STD_LOGIC;

  -- interne mikrooperacije
  -- indikacija izvrsenja mikroalgoritma
  SIGNAL sX      : STD_LOGIC;
  -- adresa naredne mikroinstrukcije
  SIGNAL sADR    : STD_LOGIC_VECTOR(2 DOWNTO 0);
  -- adresiranje multipleksera za kotrolu adresnog registra
  SIGNAL sG      : STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN

  -- multiplekser za kontrolu adresnog registra
  --   izlaz je signal za dozvolu paralelnog upisa
  --   adrese sledece mikroinstrukcije
  PROCESS (sG, iE, iSIGN) BEGIN
    CASE sG IS
      WHEN "00"   => sLOAD <= '0';
      WHEN "01"   => sLOAD <= '1';
      WHEN "10"   => sLOAD <= iSIGN;
      WHEN "11"   => sLOAD <= iE;
      WHEN OTHERS => sLOAD <= '0';
    END CASE;
  END PROCESS;

  -- signal dozvole uvecanja adresnog registra za 1
  sINCR <= NOT sLOAD;

  -- adresni registar
  PROCESS (iCLK, inRESET) BEGIN
    IF (inRESET = '0') THEN
      sADR_REG <= "000"; -- postavljanje pocetnog stanja, asinhrono
    ELSIF (iCLK'EVENT AND iCLK = '1') THEN
      -- pocetak izvrsenja mikroalgoritma
      IF (sX = '1') THEN
        -- paralelni upis
        IF (sLOAD = '1') THEN
          IF (iQA = '1') THEN    -- sabiranje
            sADR_REG <= "010";
          ELSIF (iQS = '1') THEN -- oduzimanje
            sADR_REG <= "001";
```

```
          END IF;
        END IF;
      ELSE
      -- izvrsenje mikroalgoritma je u toku
        -- paralelni upis
        IF (sLOAD = '1') THEN
          sADR_REG <= UNSIGNED(sADR);
        END IF;

        -- uvecanje adrese za jedan
        IF (sINCR = '1') THEN
          sADR_REG <= sADR_REG + 1;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- mikroprogramska memorija
  eUPROG_ROM: LPM_ROM
  GENERIC MAP (
    LPM_WIDTH           => 13,
    LPM_WIDTHAD         =>  3,
    LPM_ADDRESS_CONTROL => "UNREGISTERED",
    LPM_OUTDATA         => "UNREGISTERED",
    LPM_FILE            => "uprog_uj2.mif"
  )
  PORT MAP (
    address => STD_LOGIC_VECTOR(sADR_REG),
    q       => sINSTRUCTION
  );

  -- izdvajanje mikrooperacija iz adresirane mikroinstrukcije
  sG       <= sINSTRUCTION(1 DOWNTO 0);
  sADR     <= sINSTRUCTION(4 DOWNTO 2);
  oZ       <= sINSTRUCTION(5);
  oY       <= sINSTRUCTION(6);
  oL       <= sINSTRUCTION(7);
  oC_IN    <= sINSTRUCTION(8);
  oALJ_SEL <= sINSTRUCTION(11 DOWNTO 9);
  oX       <= sINSTRUCTION(12);
  -- interni signal za kontrolu adresnog registra
  sX       <= sINSTRUCTION(12);

END ARH_uPROG_UJ;
```

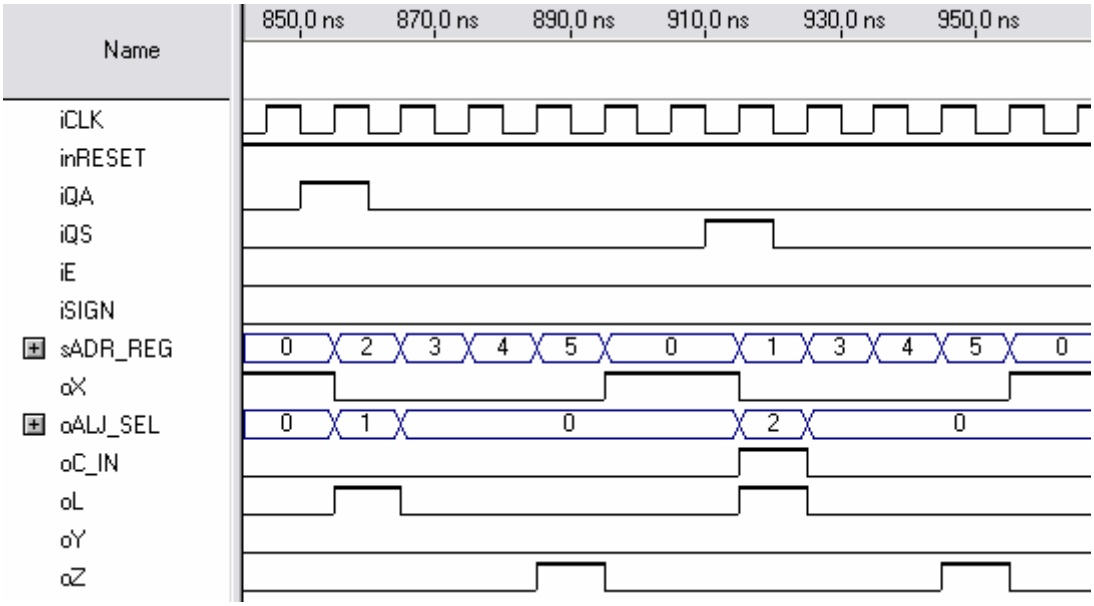
Instanca mikroprogramske memorije u prethodnom VHDL kodu je prikazana masnim slovima. Za definiciju inicijalnog sadržaja ROM memorije je iskorišćena datoteka u `.mif` formatu (*Memory Initialization File*) sa imenom `uprog_uj2.mif`. Ova datoteka je u standardnom ASCI formatu i njena struktura je izuzetno jednostavna. Potrebno je navesti širinu magistrale podataka (`WIDTH`) i broj reči u smeštenih memoriji (`DEPTH`). Vrednost adrese i podatka, `ADDRESS_RADIX` i `DATA_RADIX`, može biti predstavljena u nekoliko standardnih formata: `BIN` (binarni), `DEC` (decimalni), `HEX` (heksadecimalni), `OCT` (oktalni) ili `UNS` (decimalni neoznačeni).

Izgled ove datoteke na primeru rešenja ovog zadatka je sledeći:

```
WIDTH=13;
DEPTH=8;

ADDRESS_RADIX=UNS;
DATA_RADIX=BIN;

CONTENT BEGIN
    0   :   1000000000001;
    1   :   0010110001101;
    2   :   0001010001101;
    3   :   0000000011011;
    4   :   0000000011110;
    5   :   0000000100001;
    6   :   0000000010110;
    7   :   0000001000001;
END;
```

Slika 8.69 prikazuje vremenske dijagrame rada realizovane upravljačke jedinice. Prikazan je primer sabiranja i oduzimanja dva četvorobitna broja: A=1001 i B=0010. U ovom slučaju za oba slučaja nema prekoračenja, tj. V=0.

Sabiranje se vrši na sledeći način: izvršava se najpre prva instrukcija. Pošto je q_a=1, program vrši grananje i izvršava se mikroinstrukcija na adresi 2. Ovde signal L postaje 1, jer se vrši upis u akumulator, a zatim se učitava adresa 011. Tu se proverava da li je E=1. Pošto nije, izvršava se sledeća instrukcija. Sada se proverava da li je S=1, a kako ni ovo nije ispunjeno, izvršava se instrukcija resetovanja memorijskog elementa V, pa je prema tome, u ovoj instrukciji Z=1. Nakon toga vrši se bezuslovan skok na prvu instrukciju, gde se ostaje do pojave signala q_s ili q_a . Na sličan način vrši se i oduzimanje, s tom razlikom, da se umesto mikroinstrukcije 2, izvršava mikroinstrukcija 1.



Slika 8.69: Vremenski dijagrami rada UJ

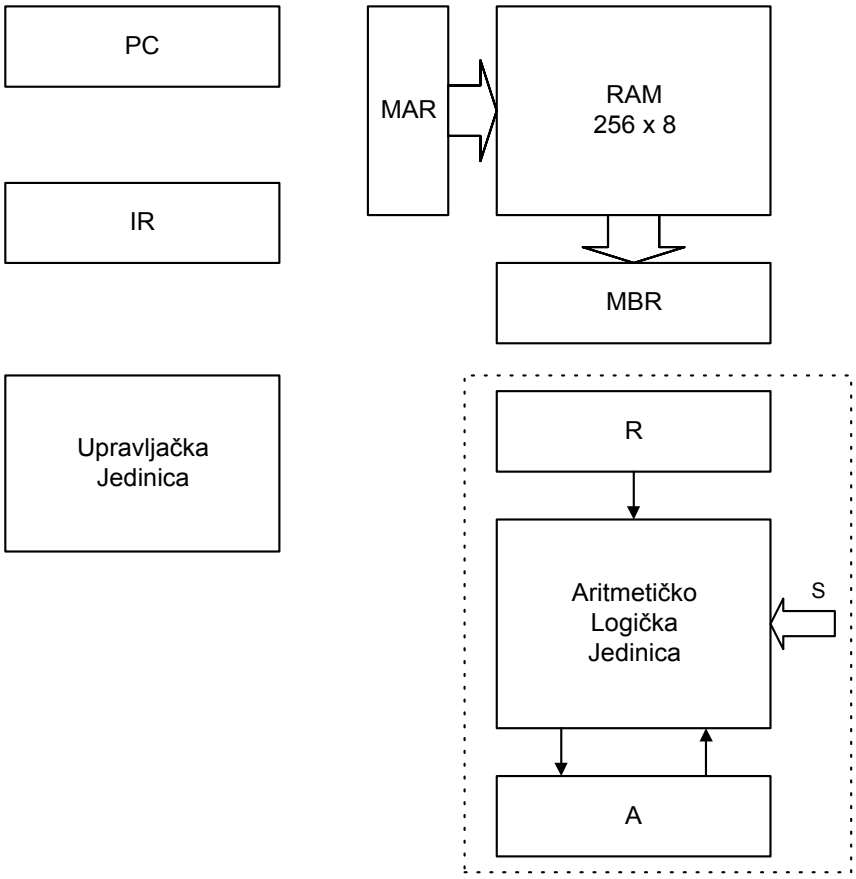
REGISTRI PROCESORA

8.16 ZADATAK:

Slika 8.70 prikazuje konfiguraciju procesora opšte namene. Nacrtati detaljnu šemu procesora i projektovati UJ na bazi PLA, za realizaciju instrukcija koje prikazuje Tabela 8.23. Svi registri su osmobitni.

Kratak opis instrukcija:

- **LDA R**: sadržaj registra R se smešta u akumulator
- **LDA adr**: u akumulator se smešta sadržaj memorijske lokacije **adr**
- **LDR oper**: smeštanje operanda oper u registar R
- **CP R**: poređenje sadržaja registara A i R
  (ne menja se sadržaj registara već samo statusne bite)



Slika 8.70: Konfiguracija procesora opšte namene

	Kod instrukcije	Mnemonik	Funkcija
q_1	0000 0001	LDA R	A ← R
q_2	0000 0010	LDA adr	A ← M[adr]
q_3	0000 0011	LDR operand	R ← operand
q_4	0000 0100	CP R	A:R (poređenje)

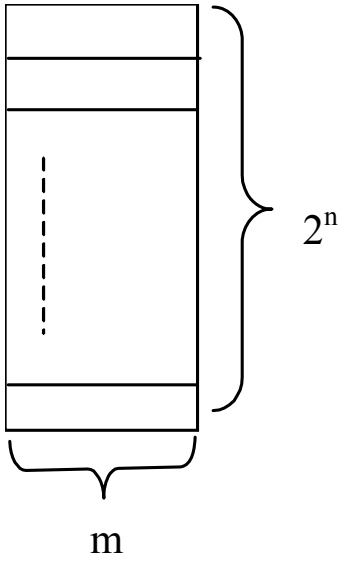
Tabela 8.23: Pregled instrukcija koje izvršava procesor

REŠENJE:

Opis skupa registara unutar procesora:

Kao što prikazuje Slika 8.70, procesor se sastoji iz:

1. **PC** (***P**rogram **C**ounter*, programski brojač) ukazuje na adresu naredne instrukcije. Pre početka izvršenja programa sadržaj ovog registra dobija vrednost početne adrese programa. Kako instrukcije programa zauzimaju kontinualne adrese PC uvećava svoj sadržaj za 1 svaki put kada se prihvati nova instrukcija, pokazujući na adresu naredne instrukcije. Ovaj redosled se prekida u slučaju izvršenja instrukcije skoka.
2. **IR** (***I**nstruction **R**egistar* – Registar Instrukcija) preuzima kod instrukcije iz memorijskog baznog registra (npr. ako se izvršava instrukcija q_1, sadržaj ovog registra je 00000001). Upravljačka jedinica formira upravljačke signale na osnovu sadržaja ovog registra.
3. **UJ** (***U**pravljačka **J**edinica*) generiše odgovarajuće upravljačke signale na osnovu koda instrukcije.
4. Memorija služi za smeštanje programa (niza instrukcija) i drugih podataka (operanada). Organizovana je kao skup od 2^n reči od kojih je svaka dužine m (Slika 8.71).



Slika 8.71: Organizacija memorije

5. **MAR** (***M**emory **A**ddress **R**egister* - Memorijski Adresni Registar) sadrži adresu memorijske lokacije kojoj se pristupa.
6. **MBR** (***M**emory **B**ase **R**egister* - Memorijski Bazni Registar) služi za prihvat podataka koji su upravo očitani iz memorije ili koji treba da se upišu u memoriju. Osnovna uloga ovog registra je da zajedno sa MAR registrom izoluju spoljnju magistralu od interne magistrale procesora.

7. **ALJ** (**A**ritmetičko **L**ogička **J**edinica), registar R i registar A (akumulator) sačinjavaju izvršni blok procesora. U okviru ovog bloka obavljaju se sve operacije nad podacima koji su smešteni u registrima.

Opis postupka sinteze UJ i šema procesora

Osnovna funkcija procesora je izvršavanje niza instrukcija koje se nalaze u operativnoj memoriji. Instrukcioni ciklus se definiše nizom mikrooperacija koje odgovaraju obradi jedne instrukcije preuzete iz operativne memorije i može se podeliti u prihvatni ciklus i ciklus izvršenja. U toku prihvatnog ciklusa iz memorije se preuzima odgovarajući kod instrukcije.

Izvršenje svake od instrukcija započinje fazom prihvata instrukcije (*fetch*). Tokom ove faze procesor prvo treba da sadržaj programskog brojača (PC) smesti u memorijski adresni registar (MAR). Nakon toga treba da pročita sadržaj memorijske lokacije na koju pokazuje MAR i uveća sadržaj programskog brojača. Na kraju ove faze procesor sadržaj memorijskog baznog registra prebacuje u registar instrukcija (IR). Vidi se da se ova faza svodi na realizaciju 3 mikrooperacije:

t_0 : MAR ← PC
t_1 : MBR ← M[MAR]; PC ← PC + 1
t_2: IR ← MBR

Na osnovu oznaka t_0, t_1 i t_2 zaključuje se da je za realizaciju ove faze potrebno tri takta.

U fazi izvršenja instrukcije prvo se vrši dekodiranje. Dekodiranje instrukcije vrši UJ generišući upravljačke signale potrebne za izvršenje odgovarajućih mikrooperacija. Nakon dekodiranja vrši se preuzimanje operanda iz memorije u slučaju da data instrukcija ima operande, a zatim sledi izvršavanje mikrooperacija definisanih instrukcijom.

Instrukcija q1 (LDA R) prebacuje sadržaj registra R u registar A. Za realizaciju ove instrukcije dovoljna je jedna mikrooperacija:

q_1t_3: A ← R; G ← 0,

što znači da se u vremenskom intervalu t_3 instrukcije q_1 desi prebacivanje sadržaja registra R u registar A i da se registar G postavi na nulu. Registar G je brojač taktova potrebnih za izvršenje instrukcije, i njegovim postavljanjem na nulu se ukazuje na kraj izvršenja tekuće instrukcije.

Instrukcija q_2 (LDA adr) prebacuje sadržaj memorijske lokacije čija je adresa smeštena uz kod instrukcije, u akumulator. Ova instrukcija zauzima 2 bajta memorijskog prostora: prvi bajt je rezervisan za kod instrukcije, a drugi za adresu operanada koji treba smestiti u A. Na početku izvršenja instrukcije, PC sadrži adresu lokacije na kojoj se nalazi kod instrukcije. U toku faze prihvata sadržaj registra PC se uvećava za jedan, tako da sadrži adresu lokacije na kojoj je adresa

operanda koji treba smestiti u registar A. Mikrooperacije koje treba realizovati nakon faze prihvata instrukcije su:

q_2t_3: MAR ← PC;
q_2t_4: MBR ← M[MAR]; PC ← PC+1; preuzimanje adrese podatka
q_2t_5: MAR ← MBR;
q_2t_6: MBR ← M[MAR]; preuzimanje podatka
q_2t_7: A ← MBR; G ← 0;

Instrukcija q_3 (LDA oper) prebacuje operand, koji je smešten uz kod instrukcije, u registar R. Kao i kod intsrukcije q_2, i ova instrukcija zauzima 2 bajta memorijskog prostora: prvi bajt služi za smeštanje koda instrukcije q_3, a drugi za smeštanje operanda:

q_3t_3: MAR ← PC;
q_3t_4: MBR ← M[MAR]; PC ← PC+1;
q_3t_5: R ← MBR; G ← 0;

Instrukcija q_4 (CP R) poredi sadržaj registara A i R. Rezultat tog poređenja, koje se izvodi oduzimanjem, smešta se u statusni registar gde se na osnovu stanja statusnih bita zaključuje o rezultatu poređenja. Za realizaciju ove instrukcije potrebna je jedna mikrooperacija:

q_4t_3: A - R; G ← 0;

Da bi se mogla nacrtati detaljna blok šema procesora, potrebno je grupisati sve mirooperacije po registrima, sa ciljem sagledavanja svih potrebnih upravljačkih signala i putanja podataka.

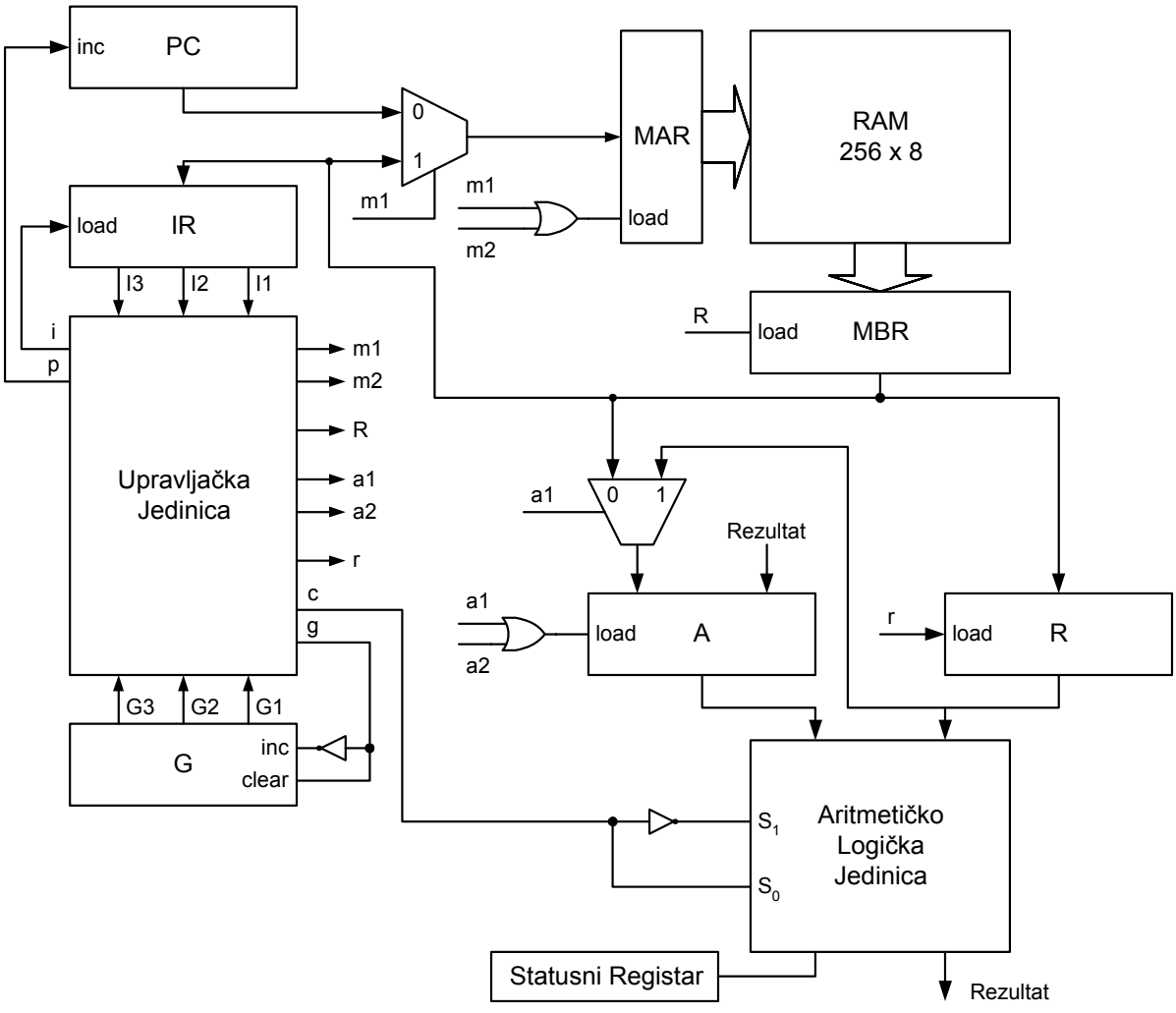
Memorija:	MBR ← M[MAR]	$R = t_1 + q_2t_4 + q_3t_4 + q_2t_6$
MAR:	MAR ← PC	$m1 = t_0 + q_2t_3 + q_3t_3$
	MAR ← MBR	$m2 = q_2t_5$
PC:	PC ← PC + 1	$p = t_1 + q_2t_4 + q_3t_4$
IR:	IR ← MBR	$i = t_2$
G:	G ← 0	$g = q_1t_3 + q_2t_7 + q_3t_5 + q_4t_3$
R:	R ← MBR	$r = q_3t_5$
A:	A ← R	$a1 = q_1t_3$
	A ← MBR	$a2 = q_2t_7$
ALJ:	A – R	$c = q_4t_3$

Na osnovu ovih informacija se može nacrtati detaljna šema procesora (Slika 8.72).

Prilikom projektovanja UJ treba imati u vidu sledeće:

1. Ulazni signali UJ su:
   - kod instrukcije koja se izvršava (smešten u IR)
   - informacija o tome do koje se mikrooperacije stiglo u izvršenju tekuće instrukcije (smeštena je u registru G)

2. Izlazni signali UJ treba da upravljaju sledećim akcijama:
   - uvećavanjem sadržaja PC za jedan (signal p)
   - punjenje MAR-a sadržajem koji je prosleđen ili iz MBR-a ili iz PC-a (signali m_1 i m_2)
   - punjenjem IR-a sadržajem MBR-a (signal i)
   - punjenjem MBR-a izlaznim signalima memorijskog bloka (signal R)
   - punjenjem registra R (signal r)
   - punjenjem registra A bilo sadržajem MBR-a , bilo sadržajem R-a (signali a_1 i a_2).
     Kada punjenje registra nije dozvoljeno, njegov sadržaj je izlaz ALJ.
   - inkrementiranjem/resetovanjem sadržaja registra G (signal g)



Slika 8.72 Detaljna blok šema procesora

Dobijanje analitičkog oblika kontrolnog signala, koji generiše UJ, će biti prikazano na primeru signala R.

Signal R upravlja punjenjem MBR-a. Punjenje MBR-a se dešava u fazi t_1 svake instrukcije, fazama t_4 i t_6 instrukcije q_2 i fazi t_4 instrukcije q_3. Na osnovu ovih informacija se dolazi do sledeće jednačine:

$$R = t_1 + q_2t_4 + q_3t_4 + q_2t_6 = t_1 + (q_2 + q_3)t_4 + q_2t_6; \qquad q2 = \overline{I_3}I_2\overline{I_1}, \qquad q3 = \overline{I_3}I_2I_1$$

$$R = t_1 + (\overline{I_3}I_2\overline{I_1} + \overline{I_3}I_2I_1)t_4 + \overline{I_3}I_2\overline{I_1}t_6 = \overbrace{t_1}^{1} + \overbrace{\overline{I_3}I_2t_4}^{2} + \overbrace{\overline{I_3}I_2\overline{I_1}t_6}^{3}$$

Sličnim postupkom se dolazi do jednačina za ostale upravljačke signale:

$$m_1 = t_0 + q_2t_3 + q_3t_3 = \overbrace{t_0}^{4} + \overbrace{t_3\overline{I_3}I_2}^{5}$$

$$m_2 = \overbrace{\overline{I_3}I_2\overline{I_1}t_5}^{6}$$

$$p = t_1 + t_4\overline{I_3}I_2$$

$$i = \overbrace{t_2}^{7}$$

$$g = \overbrace{\overline{I_3}\overline{I_2}I_1t_3}^{8} + \overbrace{I_3\overline{I_2}\overline{I_1}t_3}^{9} + \overbrace{\overline{I_3}I_2I_1t_5}^{10} + \overbrace{\overline{I_3}I_2\overline{I_1}t_7}^{11}$$

$$r = q_3t_5 = \overline{I_3}I_2I_1t_5$$

$$a_1 = q_1t_3 = \overline{I_3}\overline{I_2}I_1t_3$$

$$a_2 = q_2t_7 = \overline{I_3}I_2\overline{I_1}t_7$$

$$c = q_4t_3 = I_3\overline{I_2}\overline{I_1}t_3$$

Iz prethodnih jednačina se vidi da je minimalan broj konjunkcija koje treba realizovati 11 i da su to:

1. t_1
2. $\overline{I_3}I_2t_4$
3. $\overline{I_3}I_2\overline{I_1}t_6$
4. t_0
5. $\overline{I_3}I_2t_3$
6. $\overline{I_3}I_2\overline{I_1}t_5$
7. t_2
8. $\overline{I_3}\overline{I_2}I_1t_3$
9. $I_3\overline{I_2}\overline{I_1}t_3$
10. $\overline{I_3}I_2I_1t_5$
11. $\overline{I_3}I_2\overline{I_1}t_7$

Na osnovu prethodnih informacija, može se kreirati programska tablica PLA (6 ulaza, 10 izlaza i 11 konjunkcija). Prilikom kreiranja programske tablice koriste se sledeće oznake:

- "1" u I matrici znači da se ulazni signal u određenom proizvodu ne komplementira, "0" označava komplementiran ulaz, a "-" ulaz koji ne učestvuje u konkretnom proizvodu.
- "1" u ILI matrici označava da proizvod učestvuje u formiranju disjunktivne forme, a "-" je oznaka za proizvod koji ne učestvuje u disjunkciji.

Tabela 8.24 prikazuje programsku tablicu PLA.

	I matrica (ulazi)						ILI matrica (izlazi)									
	I_3	I_2	I_1	G_3	G_2	G_1	R	M_1	m_2	p	i	g	r	a_1	a_2	C
1.	-	-	-	0	0	1	1	-	-	1	-	-	-	-	-	-
2.	0	1	-	1	0	0	1	-	-	1	-	-	-	-	-	-
3.	0	1	0	1	1	0	1	-	-	-	-	-	-	-	-	-
4.	-	-	-	0	0	0	-	1	-	-	-	-	-	-	-	-
5.	0	1	-	0	1	1	-	1	-	-	-	-	-	-	-	-
6.	0	1	0	1	0	1	-	-	1	-	-	-	-	-	-	-
7.	-	-	-	0	1	0	-	-	-	-	1	-	-	-	-	-
8.	0	0	1	0	1	1	-	-	-	-	-	1	-	1	-	-
9.	1	0	0	0	1	1	-	-	-	-	-	1	-	-	-	1
10.	0	1	1	1	0	1	-	-	-	-	-	1	1	-	-	-
11.	1	0	1	1	1	1	-	-	-	-	-	1	-	-	1	-

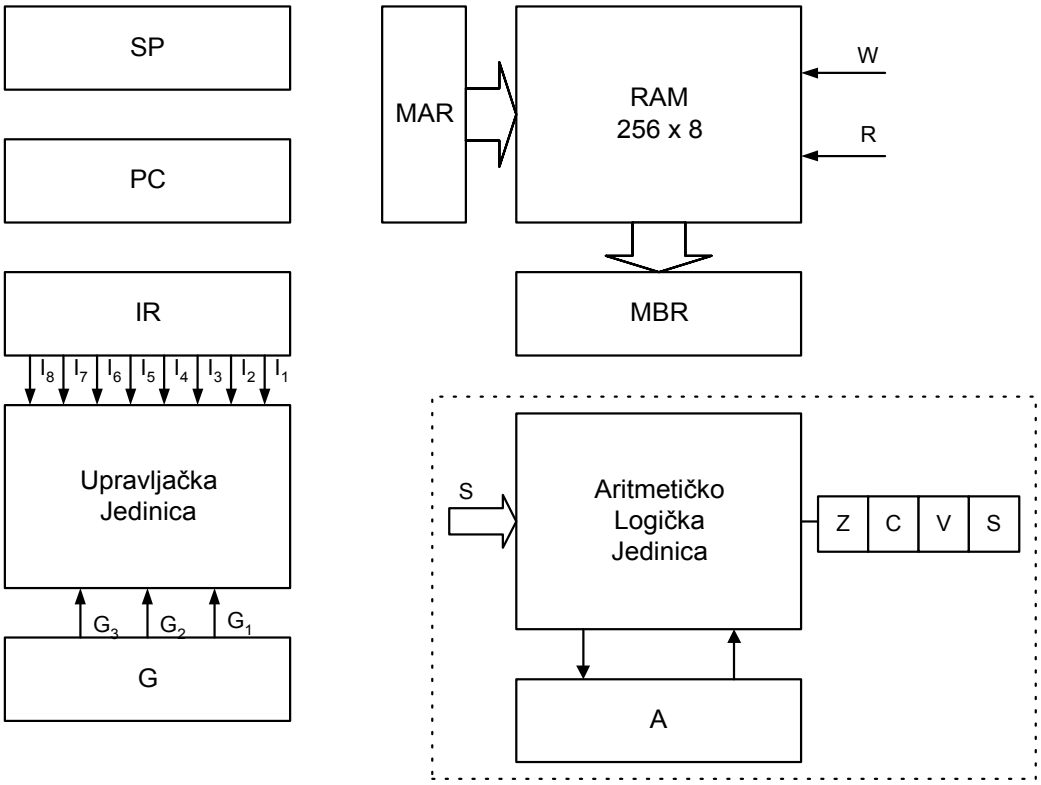
Tabela 8.24: Programska tablica PLA

8.17 ZADATAK:

Slika 8.73 prikazuje konfiguraciju procesora opšte namene. Nacrtati detaljnu šemu procesora i projektovati Upravljačku jedinicu na bazi PLA za realizaciju instrukcija koje prikazuje Tabela 8.25. Svi registri su osmobitni, izuzev registra G koji je trobitni. Koriste se celi neoznačeni brojevi.

KO	Mnemonik	Funkcija
01	PUSH A	M[SP] ← A
02	JL adr	if (C = 0) PC ← adr else PC ←
03	CP operand	A : Operand
04	AND maska	A ←A ∧ maska

Tabela 8.25: Instrukcije koje treba da realizuje UJ



Slika 8.73: Konfiguracija procesora opšte namene

Kratak opis instrukcija:

- **PUSH A**: postavljanje sadržaja registra A na vrh magazinske memorije
- **JL Adr**: ova instrukcija definiše skok u izvršavanju programa u zavisnosti od vrednosti izlaznog prenosa. Ako izlazni prenos ima vrednost 0 tada se preuzima adresa instrukcije koja će se sledeća izvršiti (Adr). Inače se PC uvećava za 1. Ova instrukcija zauzima dva bajta memorijskog prostora: jedan za kod instrukcije i drugi za adresu naredne instrukcije koja će se izvršiti u slučaju da je izlazni prenos jednak 0.
- **CP Operand**: podrazumeva poređenje sadržaja registra A i odgovarajućeg operanda. Poređenje se vrši oduzimanjem. Ova instrukcija takođe zauzima dva bajta memorijskog prostora: jedan za kod instrukcije i drugi za operand koji će se porediti sa sadržajem registra A.
- **AND maska**: ova instrukcija podrazumeva vršenje konjukcije između sadržaja registra A i maske. Sadržaj maske zavisi od toga koji biti se iz registra A izdvajaju. Kao i prethodne dve instrukcije i ova instrukcija zauzima dva bajta memorijskog prostora: jedan za kod instrukcije i drugi za masku.

REŠENJE:

Opis elemenata i registara unutar procesora:

U konfiguraciji procesora opšte namene (Slika 8.73) nalaze se sledeći blokovi:

1. **PC** (***P**rogram **C**ounter* - programski brojač) koji ukazuje na adresu naredne instrukcije. Pre početka izvršenja programa sadržaj ovog registra dobija vrednost početne adrese programa. Kako instrukcije programa zauzimaju kontinualne adrese PC uvećava svoj sadržaj za 1 svaki put kada se prihvati nova instrukcija, pokazujući na adresu naredne instrukcije. Ovaj redosled se prekida u slučaju izvršenja instrukcije skoka.
2. **SP** (***S**tack **P**ointer* - ukazivač steka) je registar koji ukazuje na vrh kontinualnih memorijskih lokacija koje čine magazinsku memoriju. Sadrži adresu elementa koji je u tom momentu na vrhu magazinske memorije.
3. **MAR** (***M**emory **A**ddress **R**egister* - Memorijski Adresni Registar) sadrži adresu memorijske lokacije kojoj se pristupa.
4. **MBR** (***M**emory **B**ase **R**egister* - Memorijski Bazni Registar) služi za prihvat podataka koji su upravo očitani iz memorije ili koji treba da se upišu u memoriju. Osnovna uloga ovog registra je da zajedno sa MAR registrom izoluju spoljnju magistralu od interne magistrale procesora.
5. **IR** (***I**nstruction **R**egistar* – Registar Instrukcija) preuzima kod instrukcije iz memorijskog baznog registra. Upravljačka Jedinica formira upravljačke signale na osnovu sadržaja ovog registra.
6. **UJ** (***U**pravljačka **J**edinica*) generiše odgovarajuće upravljačke signale na osnovu koda instrukcije.
7. **G** - brojač taktova.
8. Memorija služi za smeštanje programa (niza instrukcija) i drugih podataka (operanada).
9. **ALJ** (***A**ritmetičko **L**ogička **J**edinica*) služi za izvršavanje osnovnih aritmetičkih i logičkih operacija nad podacima.
10. **Z, C, V, S** – statusni registar čiji sadržaj čine sledeći statusni biti:
    Z – detekcija nultog sadržaja,
    C – detekcija izlaznog prenosa,
    V – detekcija prekoračenja dozvoljenog opsega brojeva,
    S – znak rezultata.

Opis postupka sinteze UJ i šema procesora:

Osnovna funkcija procesora je izvršavanje niza instrukcija koje se nalaze u operativnoj memoriji. Instrukcioni ciklus se definiše nizom mikrooperacija koje odgovaraju obradi jedne instrukcije preuzete iz operativne memorije i može se podeliti u prihvatni ciklus i ciklus izvršenja. U toku prihvatnog ciklusa iz memorije se preuzima odgovarajući kod instrukcije.

Tokom prihvata instrukcije (*fetch*) izvršava se sledeći redosled mikrooperacija:

1. Prenos adresnog dela iz instrukcionog registra (IR) u programski brojač (PC);
2. Prenos MAR ← PC, prenos sadržaja MAR na adresne linije spoljne magistrale i generisanje signala čitanja;
3. Čitanje instrukcije iz memorije i njen prenos na linije podataka spoljne magistrale;
4. Unos instrukcije u MBR, mikrooperacijom MBR ← M[MAR];
5. Prenos IR ← MBR;
6. Uvećanje sadržaja PC, PC ← PC + 1.

U fazi izvršenja instrukcije prvo se vrši dekodiranje. Dekodiranje instrukcije vrši UJ generišući upravljačke signale potrebne za izvršenje odgovarajućih mikrooperacija. Nakon dekodiranja vrši se preuzimanje operanda iz memorije u slučaju da data instrukcija ima operande, a zatim sledi izvršavanje mikrooperacija definisanih instrukcijom. Za sve instrukcije koje izvršava procesor faza prihvata instrukcije je ista i ima izgled:

t_0: MAR ← PC
t_1: MBR ← M[MAR] , PC ← PC + 1
t_2: IR ← MBR ;

dok faza izvršenja odgovarajućih instrukcija podrazumeva sledeće mikrooperacije:

instrukcija PUSH A:	q_1t_3:	SP ← SP + 1
	q_1t_4:	MAR ← SP
	q_1t_5:	MBR ← A
	q_1t_6:	M[MAR] ← MBR, G←0;
instrukcija JL adr:	q_2t_3:	MAR ← PC
	q_2t_4:	MBR ← M[MAR], PC ← PC + 1
	$q_2t_3\overline{C}$:	PC ← MBR
	q_2t_5:	G ← 0;
instrukcija CP operand:	q_3t_3:	MAR ← PC
	q_3t_4:	MBR ← M[MAR], PC ← PC + 1
	q_3t_5:	A-MBR, G ← 0
instrukcija AND maska:	q_4t_3:	MAR ← PC
	q_4t_4:	MBR ← M[MAR], PC ← PC + 1
	q_4t_5:	A ← A ∧ MBR, G ← 0

Pre definisanja programske tablice neophodno je uočiti veze između UJ tj. njenih izlaza i ostalih delova procesora. Izlazi UJ za realizaciju instrukcija koje prikazuje Tabela 8.25 imaju sledeći izgled:

Memorija:	MBR ← M[MAR]	$R = t_1 + q_2t_4 + q_3t_4 + q_4t_4$
	M[MAR] ← MBR	$W = q_1t_6$
MAR:	MAR ← PC	$m_1 = t_0 + q_2t_3 + q_3t_3 + q_4t_3$
	MAR ← SP	$m_2 = q_1t_4$
MBR:	MBR ← A	$b = q_1t_5$
IR:	IR ← MBR	$i = t_2$
SP:	SP ← SP + 1	$s = q_1t_3$
PC:	PC ← PC + 1	$p_1 = t_1 + q_2t_4 + q_3t_4 + q_4t_4$
	PC ← MBR	$p2 = q_2t_5\overline{C}$
G:	G ← 0	g = q1t6 + q2t5 + q3t5 + q4t5
ALJ:	A - MBR	s0 = q3t5
	A ← A ∧ MBR	s1 = q4t5

Broj	Proizvod	I matrica (ulazi)			ILI matrica (izlazi)											
		I_1I_0	G_2G_1G	$\overline{C}$	R	W	m_1	m_2	b	i	s	p_1	p_2	g	s_1	s_0
1	t1	-	001	-	1	-	-	-	-	-	-	1	-	-	-	-
2	q2t4	01	100	-	1	-	-	-	-	-	-	1	-	-	-	-
3	q3t4	10	100	-	1	-	-	-	-	-	-	1	-	-	-	-
4	q4t4	11	100	-	1	-	-	-	-	-	-	1	-	-	-	-
5	q1t6	00	110	-	-	1	-	-	-	-	-	-	-	1	-	-
6	t0	-	000	-	-	-	1	-	-	-	-	-	-	-	-	-
7	q2t3	01	011	-	-	-	1	-	-	-	-	-	-	-	-	-
8	q3t3	10	011	-	-	-	1	-	-	-	-	-	-	-	-	-
9	q4t3	11	011	-	-	-	1	-	-	-	-	-	-	-	-	-
10	q1t4	00	100	-	-	-	-	1	-	-	-	-	-	-	-	-
11	q1t5	00	101	-	-	-	-	-	1	-	-	-	-	-	-	-
12	t2	-	010	-	-	-	-	-	-	1	-	-	-	-	-	-
13	q1t3	00	011	-	-	-	-	-	-	-	1	-	-	-	-	-
14	q2t5	01	101	1	-	-	-	-	-	-	-	-	1	1	-	-
15	q3t5	10	101	-	-	-	-	-	-	-	-	-	-	1	-	1
16	q4t5	11	101	-	-	-	-	-	-	-	-	-	-	1	1	-

Tabela 8.26: Programska tablica UJ

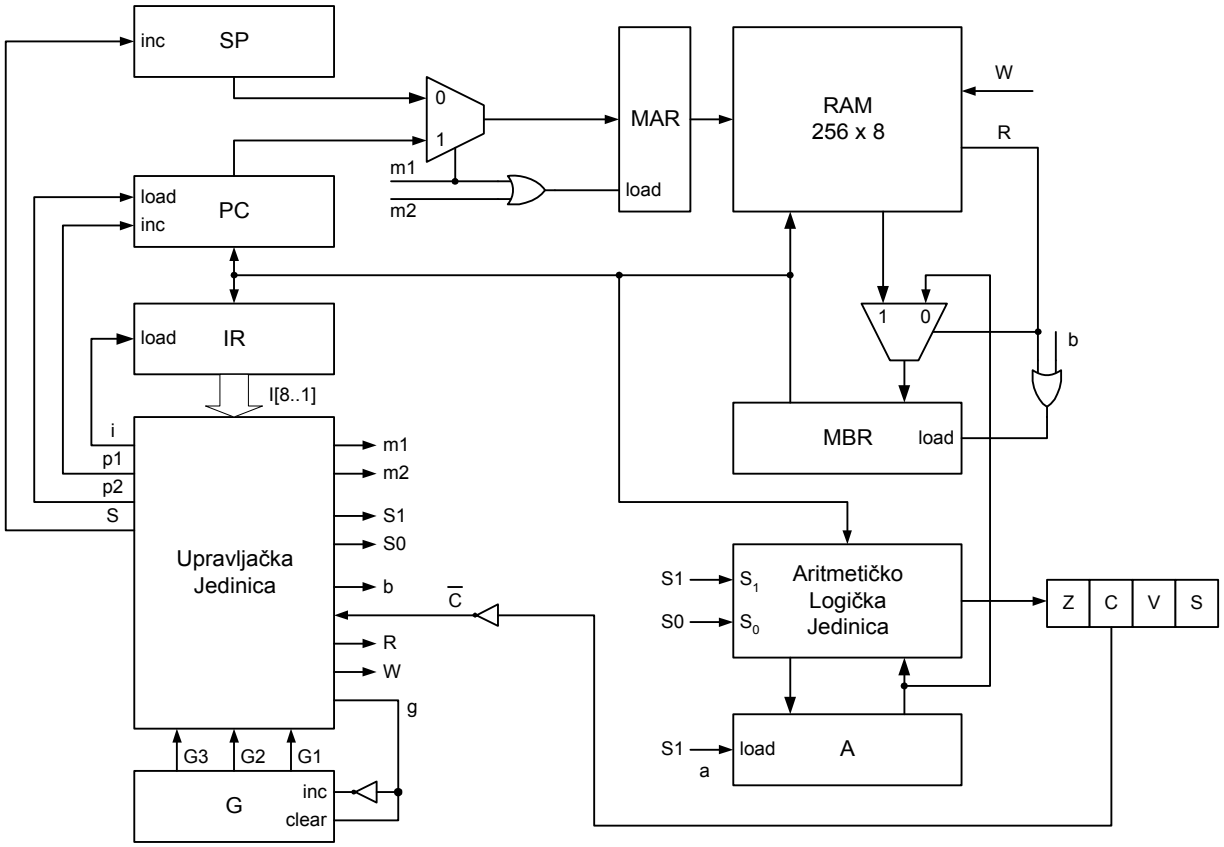
Na osnovu prethodnih informacija, može se kreirati programska tablica PLA (6 ulaza, 12 izlaza i 16 konjunkcija). Prilikom kreiranja programske tablice koriste se sledeće oznake:

- "1" u I matrici znači da se ulazni signal u određenom proizvodu ne komplementira, "0" označava komplementiran ulaz, a "-" ulaz koji ne učestvuje u konkretnom proizvodu.

– "1" u ILI matrici označava da proizvod učestvuje u formiranju disjunktivne forme, a "-" je oznaka za proizvod koji ne učestvuje u disjunkciji.

Tabela 8.26 prikazuje programsku tablicu PLA čije kolone predstavljaju redni broj proizvoda u PLA, oznaka proizvoda, kodirani ulazi u UJ i izlazi iz UJ.

Slika 8.74 prikazuje detaljnu šemu procesora sa upravljačkom jedinicom realizovanom pomoću PLA.



Slika 8.74: Detaljna šema procesora sa ALJ i UJ realizovanom pomoću PLA

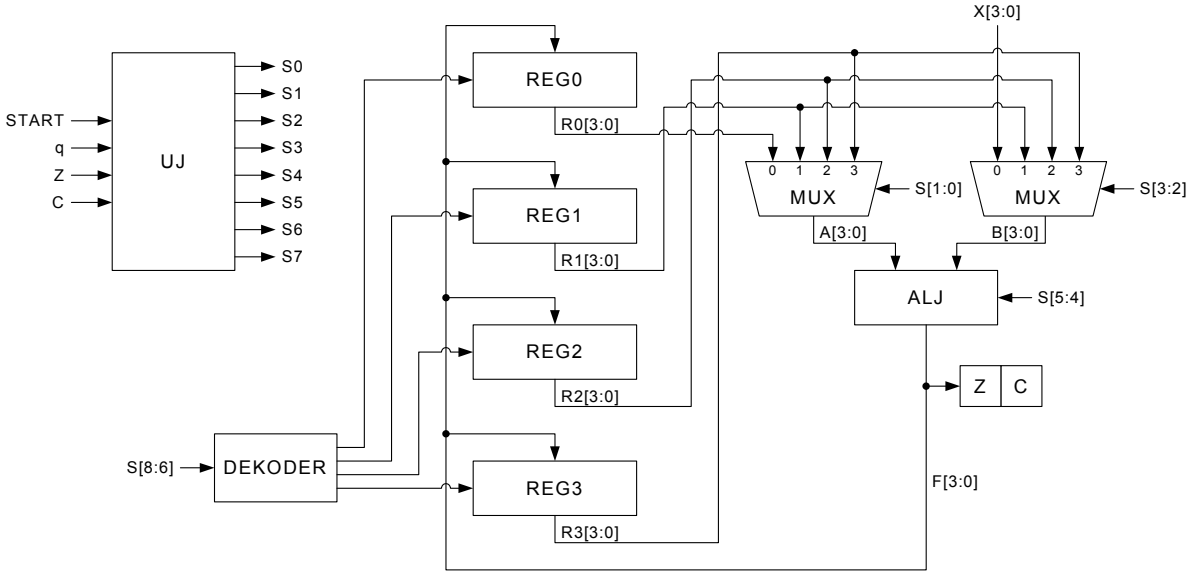
PRENOSNI PUTEVI PROCESORA

8.18 ZADATAK:

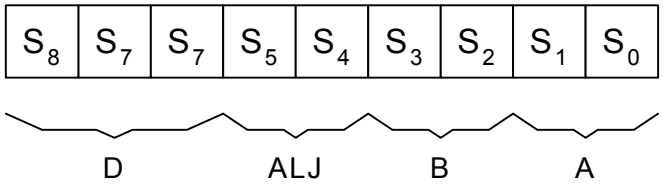
a. U VHDL jeziku za opis fizičke arhitekture izvršiti sintezu procesora sa četiri četvorobitna registra. Procesor pored ovih registara, treba da poseduje jedan dvobitni registru kome će se smeštati statusni biti Z-zero i C-carry. ALJ izvršava elementarne funkcije koje prikazuje Tabela 8.27. Arhitekturu procesora prikazuje Slika 8.75. Upravljačku reč procesora prikazuje Slika 8.76.

S_1	S_0	Operacija
0	0	F = A + B
0	1	F = A - B
1	0	F = A xnor B
1	1	F = ZERO

Tabela 8.27: Operacije koje izvršava ALJ



Slika 8.75: Blok dijagram procesora kojeg treba sintetizovati



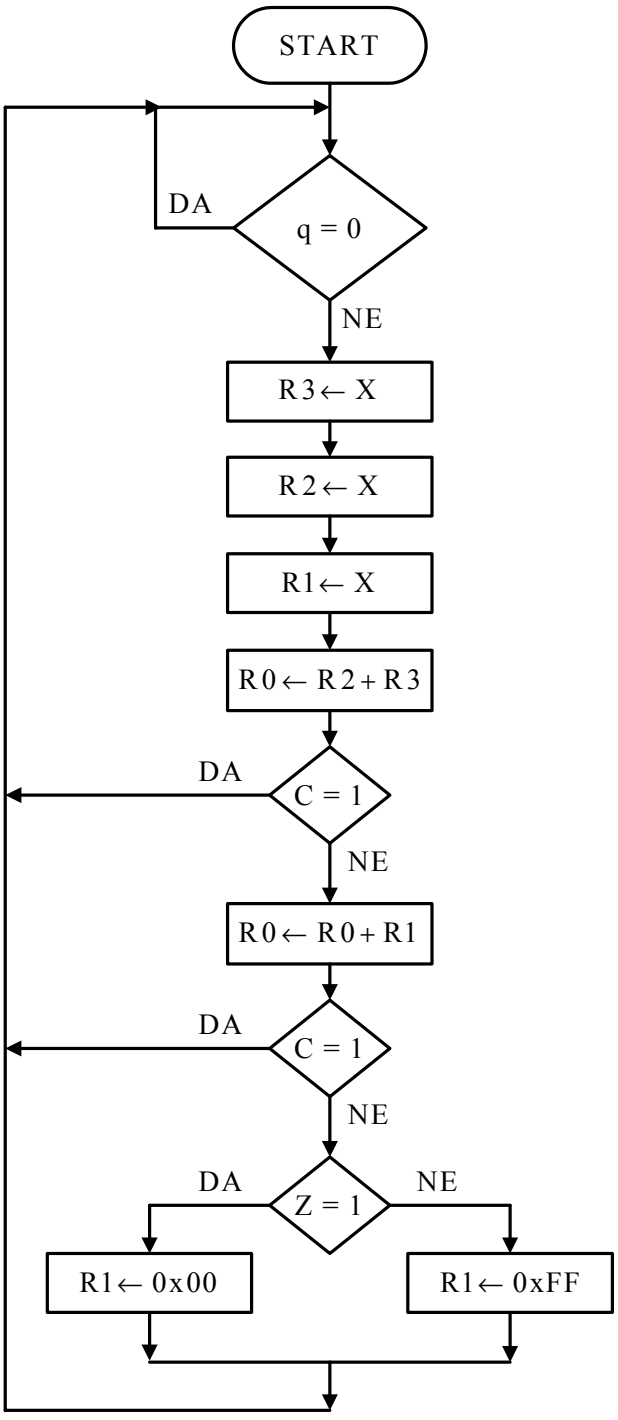
Slika 8.76: Upravljačka reč procesora

		A		B		D	
		S_0	S_1	S_2	S_3	S_6	S_7
0	0	A ← R0		B ← X		R0 ← F	
0	1	A ← R1		B ← R1		R1 ← F	
1	0	A ← R2		B ← R2		R2 ← F	
1	1	A ← R3		B ← R3		R3 ← F	

Tabela 8.28: Značenje pojedinih polja upravljačke reči

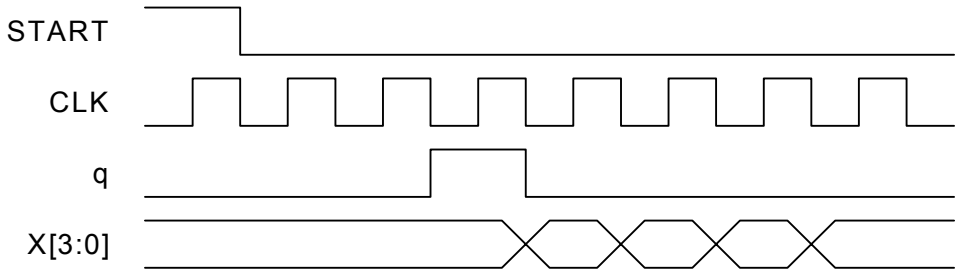
Uloga upravljačkog bita S_8 je da dozvoli proces dekodovanja (S_8=1) ili da ne dozvoli (S_8=0). U slučaju da nije dozvoljen proces dekodovanja, ne generišu se signali na izlazu dekodera, tj. nije dozvoljen upis u registre.

b. Izvršiti sintezu upravljačke jedinice koja će realizovati mikroprogram dat u obliku blok dijagrama algoritma (Slika 8.77). Sitezu izvršiti u VHDL jeziku za opis fizičke arhitekture metodom jedan memorijski element po stanju.



Slika 8.77: Blok dijagram algoritma za sintezu mikroprograma

Procesor i UJ se taktuju zajedničkim signalom, CLK, i resetuju se zajedničkim signalom, START. Voditi računa o sinhronizaciji pojedinih komponenti, ukoliko izgled signala kojima se pobuđuje procesor prikazuje Slika 8.78.



Slika 8.78: Vremenski prikaz ulaznih signala

REŠENJE:

Analizom blok dijagrama procesora, Slika 8.75, koji treba da se isprojektuje, vidi se da je potrebno izvršiti sintezu sledećih entiteta:

- registar,
- vektorski multiplekser 4 na 1,
- aritmetičko logička jedinica i
- dekoder 2 na 4 sa signalom dozvole dekodovanja.

Na kraju sledi sinteza upravljačke jedinice koja generiše upravljačke signale za izvršenje zadatog algoritma (Slika 8.77) na isprojektvanom procesoru.

REGISTAR:

Prvi realizovani modul je registar. Spregu ovog modula čine sledeći signali:

- `iCLK` takt signal,
- `iCLR` signal za brisanje sadržaja registra, aktivan na visokom nivou i sinhron sa signalom takta,
- `iCE` dozvola upisa u registar (ako je ovaj signal na visokom nivou u registar se na rastuću ivicu takt signala upisuje vrednost prisutna na ulaznom vektoru podataka `iD`),
- `iD` ulazni vektor podataka i
- `oQ` stanje registra.

Sa ciljem formiranja generalizovanog registra sa prizvoljnim brojem bita uvodi se generički parametar `pWIDTH` koji označava broj bita registra koji se instancira. Uvođenjem ovog parametra registar će imati bite sa indeksima `0` (bit najmanje važnosti) do `pWIDTH-1` (bit najveće važnosti), što ukupno čini `pWIDTH` bita. Pretpostavlja se da instancirani registar ima četiri bita.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY REG IS
  GENERIC (
    pWIDTH: integer := 4    -- pretpostavljeni broj bita je 4
  );
  PORT (
    iCLK, iCLR: IN  STD_LOGIC;
    iCE: IN  STD_LOGIC;
    iD:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
    oQ:  OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
  );
END REG;

ARCHITECTURE ARH_REG OF REG IS
  -- stanje registra
  SIGNAL sREG: STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
BEGIN
  PROCESS (iCLK) BEGIN
    IF (iCLK'EVENT AND iCLK = '1') THEN
      -- sinhrono postavljanje pocetne vrednosti
      IF (inCLR = '1') THEN
        sREG <= (OTHERS => '0');
      ELSE
        IF (iCE = '1') THEN -- upis u registar
          sREG <= iD;
        ELSE
          sREG <= sREG;
        END IF;
      END IF;
    END IF;
  END PROCESS;

  -- preslikavanje stanja registra na izlazni vektor
  oQ <= sREG;

END ARH_REG;
```

VEKTORSKI MULTIPLEKSER 4 NA 1:

Multiplekser se koristi za odabir operanada koji učestvuju u izvršenju aritmetičko/logičke operacije. Spregu ovog modula čine sledeći signali:

- `iX0` ulazni vektor sa adresom 0,
- `iX1` ulazni vektor sa adresom 1,
- `iX2` ulazni vektor sa adresom 2,
- `iX3` ulazni vektor sa adresom 3,
- `iSEL` adresni ulaz multipleksera i
- `oY` izlazni vektor.

Širina ulaznih i izlaznih vektora se određuje preko generičkog parametra `pWIDTH`. Pretpostavljena vrednost ovog parametra je 4.

```
  LIBRARY ieee;
    USE ieee.std_logic_1164.all;

  ENTITY MUX4x1 IS
    GENERIC (
      pWIDTH: integer := 4    -- pretpostavljeni broj bita je 4
    );
    PORT (
      iX0, iX1,
      iX2, iX3: IN  std_logic_vector(pWIDTH-1 DOWNTO 0);
      iSEL:     IN  std_logic_vector(1 DOWNTO 0);
      oY:       OUT std_logic_vector(pWIDTH-1 DOWNTO 0)
    );
  END MUX4x1;

  ARCHITECTURE ARH_MUX4x1 OF MUX4x1 IS BEGIN
    PROCESS (iX0, iX1, iX2, iX3, iSEL) BEGIN
      -- odredjivanje izlaznog vektora
      -- u zavisnosti od vrednosti adresnog vektora
      CASE iSEL IS
        WHEN "00"   => oY <= iX0;
        WHEN "01"   => oY <= iX1;
        WHEN "10"   => oY <= iX2;
        WHEN OTHERS => oY <= iX3;
      END CASE;
    END PROCESS;
  END ARH_MUX4x1;
```

ALJ:

Aritmetičko logička jedinica služi za izvršenje zadatih operacija (Tabela 8.27). Sprežni signali tražene ALJ su sledeći:

- iA prvi ulazni četvorobitni operand,
- iB drugi ulazni četvorobitni operand,
- iSEL adresni ulaz ALJ za odabir operacije,
- oH rezultat izvršene aritmetičko logičke operacije i
- oC izlazni prenos.

ALJ je sintetizovana pomoću vektora na način koji je ranije prikazan u poglavlju posvećenom projektovanju aritmetičko logičkih jedinica.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;

ENTITY ALJ IS PORT(
  iA, iB: IN  std_logic_vector(3 DOWNTO 0);
  iSEL:   IN  std_logic_vector(1 DOWNTO 0);
  oH:     OUT std_logic_vector(3 DOWNTO 0);
  oC:     OUT std_logic );
END ALJ;
```

```
ARCHITECTURE ARH_ALJ OF ALJ IS
  -- rezultat operacije
  -- prosiren za jedan bit gde ce se smestiti
  -- izracunati izlazni prenos
  SIGNAL sF: unsigned(4 DOWNTO 0);
BEGIN

 PROCESS(iSEL, iA, iB)
  -- operandi koji su prosireni za jedan bit
  -- zbog racunanja izlaznog prenosa
  VARIABLE vA, vB: unsigned(4 DOWNTO 0);
 BEGIN
  vA := unsigned('0' & iA); -- operand A prosiren za jedan bit
  vB := unsigned('0' & iB); -- operand B prosiren za jedan bit

  CASE iSEL(1 DOWNTO 0) IS
   WHEN "00"   => sF <= vA + vB;
   WHEN "01"   => sF <= vA + (('0' & NOT(vB(3 DOWNTO 0)))) + 1;
   WHEN "10"   => sF <= ('0' & (vA(3 DOWNTO 0) XNOR vB(3 DOWNTO 0)));
   WHEN OTHERS => sF <= "00000";
  END CASE;
 END PROCESS;

  -- rezultat ALJ
  oH <= std_logic_vector(sF(3 DOWNTO 0));
  -- signal izlaznog prenosa,
  oC <= std_logic(sF(4));

 END ARH_ALJ;
```

DEKODER 2 NA 4:

Sprežni signali dekodera su sledeći:

- iX ulazni vektor dekodera,
- iEN signal dozvole dekodovanja aktivan na visokom nivou i
- oY izlazni vektor dekodera gde je aktivan (ima vrednost jedan) samo bit na poziciji na koju ukazuje vrednost ulaznog vektora dekodera.

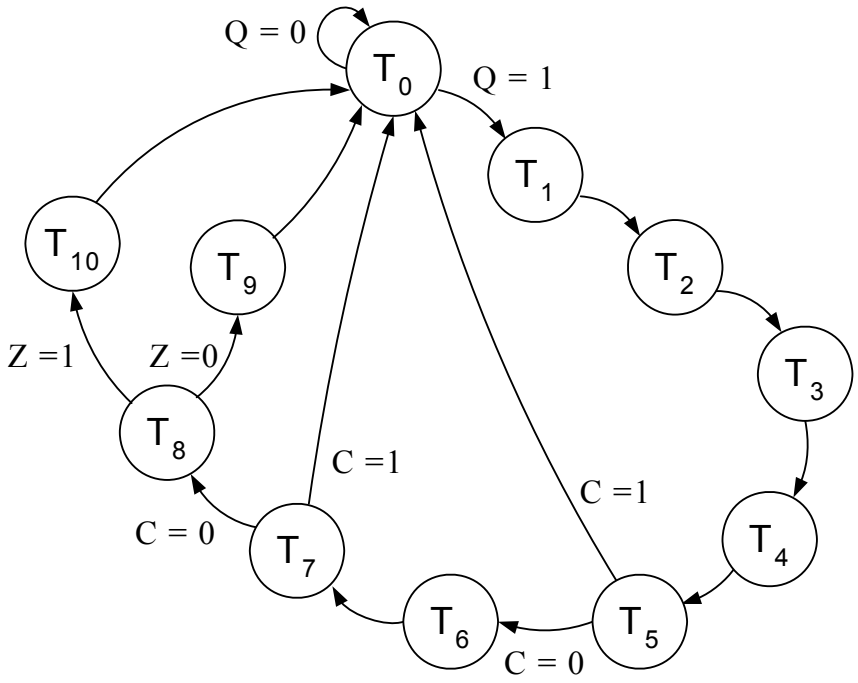
U slučaju da nije aktivan signal dozvole dekodovanja svi izlazni signali su u neaktivnom stanju, tj. imaju vrednost nula.

```
  LIBRARY ieee;
    USE ieee.std_logic_1164.all;

  ENTITY DECODER IS PORT(
    iX:  IN  std_logic_vector (1 DOWNTO 0);   -- ulaz dekodera
    iEN: IN  std_logic;                       -- signal dozvole
dekodovanja
    oY:  OUT std_logic_vector (3 DOWNTO 0) ); -- izlaz dekodera
  END DECODER;

  ARCHITECTURE ARH_DECODER OF DECODER IS BEGIN
    PROCESS (iX, iEN) BEGIN
```

```
    IF (iEN = '1') THEN
      CASE iX IS
        WHEN "00"   => oY <= "0001";
        WHEN "01"   => oY <= "0010";
        WHEN "10"   => oY <= "0100";
        WHEN "11"   => oY <= "1000";
        WHEN OTHERS => oY <= "0000";
      END CASE;
    ELSE
      oY <= "0000";
    END IF;
  END PROCESS;
END ARH_DECODER;
```

UPRAVLJAČKA JEDINICA:

Ovaj modul treba da na osnovu zadatog blok dijagrama algoritma, Slika 8.77, generiše odgovarajuće upravljačke signale za sve komponente procesora. Radi lakše implementacije formira se graf prelazaka stanja, Slika 8.79. U svakom stanju se generišu odgovarajući upravljački signali u skladu sa potrebnom operacijom u tom vremenskom trenutku.



Slika 8.79: Graf prelazaka stanja upravljačke jedinice

Automat se inicijalno nalazi u stanju T_0. U slučaju da se na ulazu pojavi impuls za početak izvršenja algoritma, `Q=1`, automat redom prelazi u stanja T_1, T_2 i T_3 gde se preuzima vrednost sa ulaznog vektora `X` i upisuje redom u registre `R3`, `R2` i `R1`. Nakon toga prelazi se u stanje T_4, gde se sabiraju vrednosti koje sadrže registri `R2` i `R3` i rezultat se upisuje u registar `R0`. U stanju T_5 se proverava da li je tokom izvršenja operacije sabiranja došlo do prekoračenja, tj. proverava se

vrednost statusnog bita C. U slučaju da je došlo do prekoračenja, upravljačka jedinica se vraća u stanje T_0 i završava se izvršenje algoritma. U suprotnom slučaju, prelazi se u stanje T_6, gde se sabiraju vrednosti registara R0 i R1, i rezultat se upisuje u registar R0. Iz stanja T_6 se prelazi u stanje T_7, gde se ponovo proverava da li je došlo do prekoračenja. Ako nema prekoračenja prelazi se u stanje T_8, dok se u suprotnom slučaju prelazi u stanje T_0 i završava se izvršenje algoritma. U stanju T_8 se proverava da li je rezultat sabiranja jednak nuli, proverom vrednosti statusnog bita Z. Ako je rezultat sabiranja jednak nuli, Z=1, tada se prelazi u stanje T_{10}, gde se u registar R1 upisuje vrednost 00_{HEX}. U suprotnom slučaju, Z=0, iz stanja T_8 se prelazi u stanje T_9 gde se u registar R1 upisuje vrednost FF_{HEX}. Iz stanja T_9 i T_{10} se prelazi u stanje T_0 čime se regularno završava izvršenje algoritma.

Posmatranjem grafa prelaza stanja upravljačke jedinice (Slika 8.79) vidi se da se u stanje T_0 prelazi iz stanja T_0 preko puta Q=0, iz stanja T_5 i T_7 uz uslov C=1, i direktno iz stanja T_9 i T_{10}. Slično, u stanje T_1 se prelazi iz T_0 uz uslov Q=1, u stanje T_2 bezuslovno iz stanja T_1, i tako redom. Imajući u vidu osobinu D flip-flopova da prenose informaciju prisutnu na ulazu, disjunkcijom puteva iz kojih se prelazi u neko naredno stanje, dobija se jednačina pobude svakog D flip-flopa koji je dodeljen stanju. Prema tome, uz pretpostavku da se izlazi D flip-flopova označe sa T_i, i=0, 1 ,..., 7 jednačine prelazaka stanja su sledeće:

$$D_0 = T_0 \cdot \overline{Q} + T_5 \cdot C + T_7 \cdot C + T_9 + T_{10}$$
$$D_1 = T_0 \cdot Q$$
$$D_2 = T_1$$
$$D_3 = T_2$$
$$D_4 = T_3$$
$$D_5 = T_4$$
$$D_6 = T_5 \cdot \overline{C}$$
$$D_7 = T_6$$
$$D_8 = T_7 \cdot \overline{C}$$
$$D_9 = T_8 \cdot \overline{Z}$$
$$D_{10} = T_8 \cdot Z$$

Sa ciljem formiranja jednačina izlaza, formira se tablica za operatore, T_0 do T_{10}, kao vrste u kojima su navedene mikrooperacije koje se izvršavaju kao i upravljački signali koji su u tom momentu aktivni. Pošto ALJ iz zadatka nema mogućnost propuštanja operanda B koja bi se iskoristila za upis ulazne vrednosti X u registre, ovu operaciju je moguće izvšiti mikrooperacijom sabiranja ulazne vrednosti X i nekog registra čija je vrednost 0. Zbog toga se u stanju T_0 u registar R0 upisuje vrednost nula, a u stanjima T_1, T_2 i T_3 izvršava operacija sabiranja X+R0 i rezultat upisuje redom u registre R4, R3 i R2. Za upis vrednosti FF_{HEX} u registar R1 je iskorišćena operacija R1 xnor R1. To prikazuje (Tabela 8.29).

		Izlazni signali								
		S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
T_0	početno stanje	1	0	0	1	1	0	0	0	0
T_1	$R_3 \leftarrow X$	1	1	1	0	0	0	0	0	0
T_2	$R_2 \leftarrow X$	1	1	0	0	0	0	0	0	0
T_3	$R_1 \leftarrow X$	1	0	1	0	0	0	0	0	0
T_4	$R_0 \leftarrow R_2 + R_3$	1	0	0	0	0	1	0	1	1
T_5	provera C	0	0	0	0	0	0	0	0	0
T_6	$R_0 \leftarrow R_0 + R_1$	1	0	0	0	0	0	1	0	0
T_7	provera C	0	0	0	0	0	0	0	0	0
T_8	provera Z	0	0	0	0	0	0	0	0	0
T_9	$R_1 \leftarrow 0xFF$	1	0	1	1	0	0	1	0	1
T_{10}	$R_1 \leftarrow 0x00$	1	0	1	1	1	0	0	0	0

Tabela 8.29: Tabela izlaznih signala UJ

Tabela 8.17 definiše jednačine izlaza upravljačkog automata kao:

$S_0 = T_4 + T_9$ $\quad S_5 = T_0 + T_9 + T_{10}$

$S_1 = T_4$ $\quad S_6 = T_1 + T_3 + T_9 + T_{10}$

$S_2 = T_6 + T_9$ $\quad S_7 = T_1 + T_2$

$S_3 = T_4$ $\quad S_8 = T_0 + T_1 + T_2 + T_3 + T_4 + T_6 + T_9 + T_{10}$

$S_4 = T_0 + T_{10}$

UJ radi tako što je u jednom trenutku vremena samo jedan flip-flop aktivan, a svi ostali imaju nulti izlaz. Sledeća rastuća ivica takta aktivira samo flip-flop na čijem je D ulazu 1, svi ostali se brišu. Drugim rečima, ulazne funkcije flip-flopova su međusobno isključive i samo jedan flip-flop može biti aktivan. Prelaz u naredno stanje iz tekućeg je funkcija tekućeg T_i koje je 1 i nekih ulaznih uslova. Naredno stanje počinje po brisanju prethodnog. Da bi ovaj upravljački automat mogao da počne da radi, mora se inicirati početno stanje (T_0) i to tako što će se flip-flop koji reprezentuje to stanje staviti u aktivno stanje.

Upravljački automat u ovom slučaju ima sledeće sprežne signale:

- `iCLK` takt signal,
- `iRESET` signal za postavljanje upravljačke jedinice u inicijalno, aktivan na visokom nivou i sinhron sa signalom takta,
- `iQ` impuls za početak izvršenja algoritma,
- `iZ` stanje statusnog bita koji ukazuje da je rezultat prethodne izvršene operacije jednak nuli (`oZ=1`). U suprotnom je `oZ=0`.
- `iC` stanje statusnog bita koji ukazuje na izlazni prenos prethodno izvršene operacije.

– oS vektor od ukupno 9 upravljačkih signala koji čine upravljačku reč procesora, Slika 8.76.

Stanja upravljačke jedinice su reprezentovana sa signalima sT0 do sT10, dok su jednačine prelazaka stanja predstavljene signalima sD0 do sD10.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

ENTITY UJ IS PORT(
  iCLK, iRESET: IN std_logic;
  iQ, iZ, iC:   IN std_logic;
  oS: OUT std_logic_vector(8 DOWNTO 0) );
END UJ;

ARCHITECTURE ARH_UJ of UJ IS
  -- deklaracija signala koji odredjuju tekuce stanje UJ
  -- (izlazi flip-flopova)
  SIGNAL sT0, sT1, sT2, sT3, sT4, sT5,
         sT6, sT7, sT8, sT9, sT10 : STD_LOGIC;

  -- deklaracija signala koji odredjuju naredno stanje UJ
  -- (ulazi flip-flopova)
  SIGNAL sD0, sD1, sD2, sD3, sD4, sD5,
         sD6, sD7, sD8, sD9, sD10 : STD_LOGIC;
BEGIN
  -- kombinaciona mreza koja na osnovu trenutnog stanja
  -- i vrednosti ulaza generise kod narednog stanja
  sD0  <= (NOT(iQ) AND sT0) OR (iC AND sT5) OR
          (iC AND sT7) OR sT9 OR sT10;
  sD1  <= iQ AND sT0;
  sD2  <= sT1;
  sD3  <= sT2;
  sD4  <= sT3;
  sD5  <= sT4;
  sD6  <= NOT(iC) AND sT5;
  sD7  <= sT6;
  sD8  <= NOT(iC) AND sT7;
  sD9  <= NOT(iZ) AND sT8;
  sD10 <= iZ AND sT8;

  -- flip-flopovi za smestanje koda
  -- trenutnog stanja; postavljanje pocetnog
  -- stanja je sinhrono sa signalom takta
  PROCESS (iCLK) BEGIN
    IF (iCLK'event and iCLK='1') THEN
      IF (iRESET = '1') then  -- sinhroni reset
        sT0  <= '1';  sT1 <= '0';
        sT2  <= '0';  sT3 <= '0';
        sT4  <= '0';  sT5 <= '0';
        sT6  <= '0';  sT7 <= '0';
        sT8  <= '0';  sT9 <= '0';
        sT10 <= '0';
      ELSE
        sT0  <= sD0;  sT1 <= sD1;
        sT2  <= sD2;  sT3 <= sD3;
```

```
        sT4  <= sD4;  sT5 <= sD5;
        sT6  <= sD6;  sT7 <= sD7;
        sT8  <= sD8;  sT9 <= sD9;
        sT10 <= sD10;
      END IF;
    END IF;
  END PROCESS;

   -- odredjivanje vrednosti izlaznih signala
   -- na osnovu trenutnog stanja
   oS(0) <= sT4 OR sT9;
   oS(1) <= sT4;
   oS(2) <= sT6 OR sT9;
   oS(3) <= sT4;
   oS(4) <= sT0 OR sT10;
   oS(5) <= sT0 OR sT9 OR sT10;
   oS(6) <= sT1 OR sT3 OR sT9 OR sT10;
   oS(7) <= sT1 OR sT2;
   oS(8) <= sT0 OR sT1 OR sT2 OR sT3 OR sT4 OR sT6 OR sT9 OR sT10;

 END ARH_UJ;
```

PROCESOR:

U ovom modulu se instanciraju svi prethodno opisani moduli i međusobno povezuju u skladu sa blok dijagramom procesora, Slika 8.75.

Prvo je potrebno formirati pakovanje sa opisom sprega svih modula koji se instanciraju:

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

PACKAGE CPU_PKG IS
  COMPONENT REG
    GENERIC (
      pWIDTH: integer := 4
    );
    PORT (
      iCLK, iCLR: IN  STD_LOGIC;
      iCE: IN  STD_LOGIC;
      iD:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
      oQ:  OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
    );
  END COMPONENT;

  COMPONENT MUX4x1 IS
    GENERIC (
      pWIDTH: integer := 4
    );
    PORT (
      iX0, iX1,
      iX2, iX3: IN  std_logic_vector(pWIDTH-1 DOWNTO 0);
      iSEL:     IN  std_logic_vector(1 DOWNTO 0);
      oY:       OUT std_logic_vector(pWIDTH-1 DOWNTO 0)
    );
  END COMPONENT;
```

```
  COMPONENT DECODER IS
    PORT(
      iX:  IN  std_logic_vector (1 DOWNTO 0);
      iEN: IN  std_logic;
      oY:  OUT std_logic_vector (3 DOWNTO 0)
    );
  END COMPONENT;

  COMPONENT ALJ IS
    PORT(
      iA, iB: IN  std_logic_vector(3 DOWNTO 0);
      iSEL:   IN  std_logic_vector(1 DOWNTO 0);
      oH:     OUT std_logic_vector(3 DOWNTO 0);
      oC:     OUT std_logic
    );
  END COMPONENT;

  COMPONENT UJ
    PORT (
      iCLK, iRESET: IN std_logic;
      iQ, iZ, iC:   IN std_logic;
      oS: OUT std_logic_vector(8 DOWNTO 0)
    );
  END COMPONENT;

END CPU_PKG;
```

Nakon formiranja pakovanja sa opisom sprega svih komponenti koje se instanciraju prelazi se na opis entiteta projektovanog procesora. Projektovani modul procesora sadrži sledeće ulazno/izlazne signale:

- `iCLK` takt signal,
- `iSTART` signal za brisanje sadržaja svih registara i brojača, aktivan na visokom nivou i sinhron sa signalom takta,
- `iQ` impuls za početak izvršenja zadatog algoritma,
- `iX` četvorobitni ulazni vektor,
- `oH` rezultat izvršene aritmetičko logičke operacije, četvorobitni izlazni vektor,
- `oZ` stanje statusnog bita koji ukazuje da je rezultat prethodne izvršene operacije jednak nuli (`oZ=1`). U suprotnom je `oZ=0`.
- `oC` stanje statusnog bita koji ukazuje na izlazni prenos prethodno izvršene operacije.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE work.CPU_PKG.all; -- ukljucivanje pakovanja sa komponentama

ENTITY CPU IS
  PORT (
    iCLK:       IN  STD_LOGIC;
    iSTART, iQ: IN  STD_LOGIC;
    iX:         IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
```

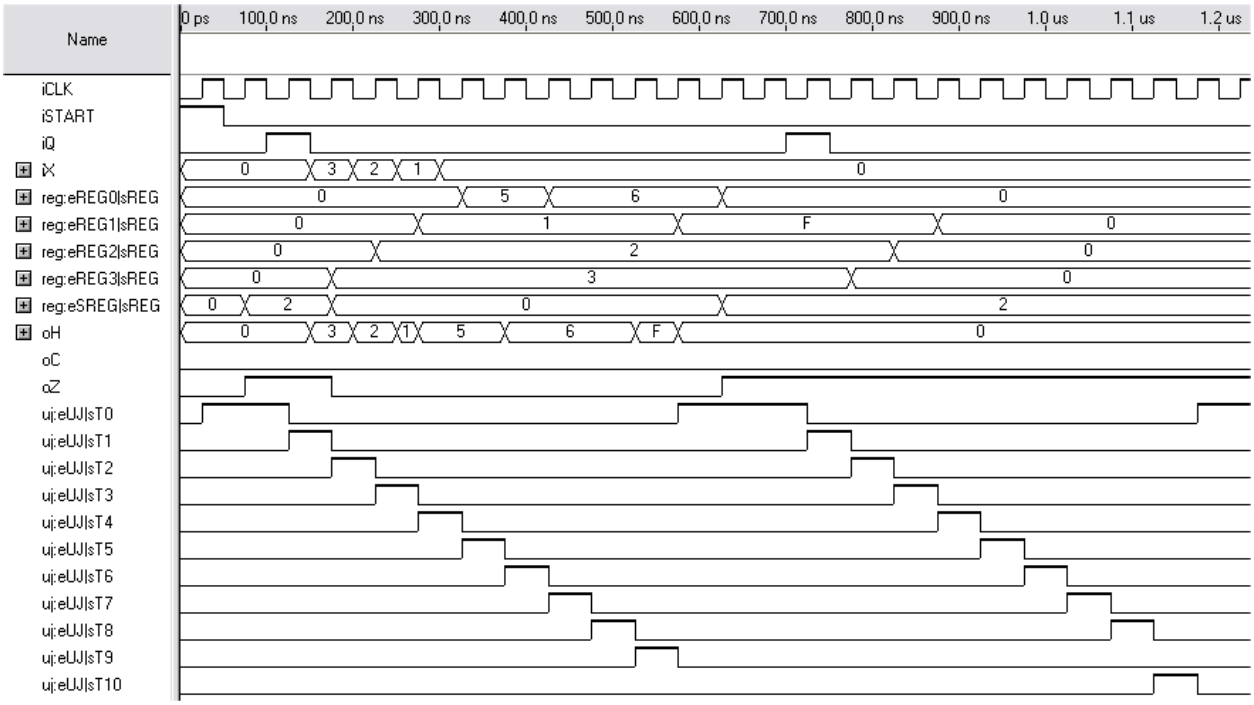
```
    oH:          OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    oZ, oC:      OUT STD_LOGIC
  );
END CPU;

ARCHITECTURE ARH_CPU OF CPU IS
  -- konstanta 1
  CONSTANT cONE: STD_LOGIC := '1';
  -- registri
  SIGNAL sR0, sR1, sR2, sR3: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- ulazni vektori u ALJ
  SIGNAL sA, sB: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- rezultat ALJ
  SIGNAL sF: STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL sC_OUT: STD_LOGIC;  -- izlazni prenos
  -- signal koji ukazuje da je rezultat ALJ operacije 0
  SIGNAL sZERO: STD_LOGIC;
  -- statusni registar
  SIGNAL sDATA: STD_LOGIC_VECTOR(1 DOWNTO 0); -- ulazni vektor
  SIGNAL sSREG: STD_LOGIC_VECTOR(1 DOWNTO 0); -- izlazni vektor

  -- statusni biti
  SIGNAL sC, sZ: STD_LOGIC;
  -- izlaz dekodera -> dozvola upisa u registre
  SIGNAL sWE: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- upravljacki signali
  SIGNAL sS: STD_LOGIC_VECTOR(8 DOWNTO 0);
BEGIN

  -- registar na adresi 0
  eREG0: REG
    GENERIC MAP (pWIDTH=>4)
    PORT    MAP (iCLK=>iCLK, iCLR=>iSTART, iCE=>sWE(0),
                 iD=>sF, oQ=>sR0);

  -- registar na adresi 1
  eREG1: REG
    GENERIC MAP (pWIDTH=>4)
    PORT    MAP (iCLK=>iCLK, iCLR=>iSTART, iCE=>sWE(1),
                 iD=>sF, oQ=>sR1);

  -- registar na adresi 2
  eREG2: REG
    GENERIC MAP (pWIDTH=>4)
    PORT    MAP (iCLK=>iCLK, iCLR=>iSTART, iCE=>sWE(2),
                 iD=>sF, oQ=>sR2);

  -- registar na adresi 3
  eREG3: REG
    GENERIC MAP (pWIDTH=>4)
    PORT    MAP (iCLK=>iCLK, iCLR=>iSTART, iCE=>sWE(3),
                 iD=>sF, oQ=>sR3);

  -- multiplekser za odabir operanda A
  eMUXA: MUX4x1
    PORT MAP(iX0=>sR0, iX1=>sR1, iX2=>sR2, iX3=>sR3,
             iSEL=>sS(1 DOWNTO 0), oY=>sA);
```

```
    -- multiplekser za odabir operanda B
    eMUXB: MUX4x1
      PORT MAP(iX0=>iX, iX1=>sR1, iX2=>sR2, iX3=>sR3,
               iSEL=>sS(3 DOWNTO 2), oY=>sB);

    -- ALJ
    eALJ: ALJ
      PORT MAP(iA=>sA, iB=>sB, iSEL=>sS(5 DOWNTO 4),
               oH=>sF, oC=>sC_OUT);

    -- formiranje signala koji ukazuje
    -- da je rezultat ALJ operacije nula
    sZERO <= NOT(sF(0) OR sF(1) OR sF(2) OR sF(3));

    -- statusni registar
    sDATA <= (sZERO & sC_OUT);  -- ulazna vrednost registra
    eSREG: REG
      GENERIC MAP (pWIDTH=>2)
      PORT    MAP (iCLK=>iCLK, iCLR=>iSTART, iCE=>cONE,
                   iD=>sDATA, oQ=>sSREG);

    -- vrednost statusnih bita
    sC <= sSREG(0);  -- carry
    sZ <= sSREG(1);  -- zero

    -- dozvola upisa u registre
    eDEKODER: DECODER
      PORT MAP(iX=>sS(7 DOWNTO 6), iEN=>sS(8), oY=>sWE);

    -- upravljacka jedinica
    eUJ: UJ
      PORT MAP (iCLK=>iCLK, iRESET=>iSTART, iQ=>iQ, iZ=>sZ,
                iC=>sC, oS=>sS);

    -- izlazni signali
    oH <= sF;
    oC <= sC;
    oZ <= sZ;
  END ARH_CPU;
```

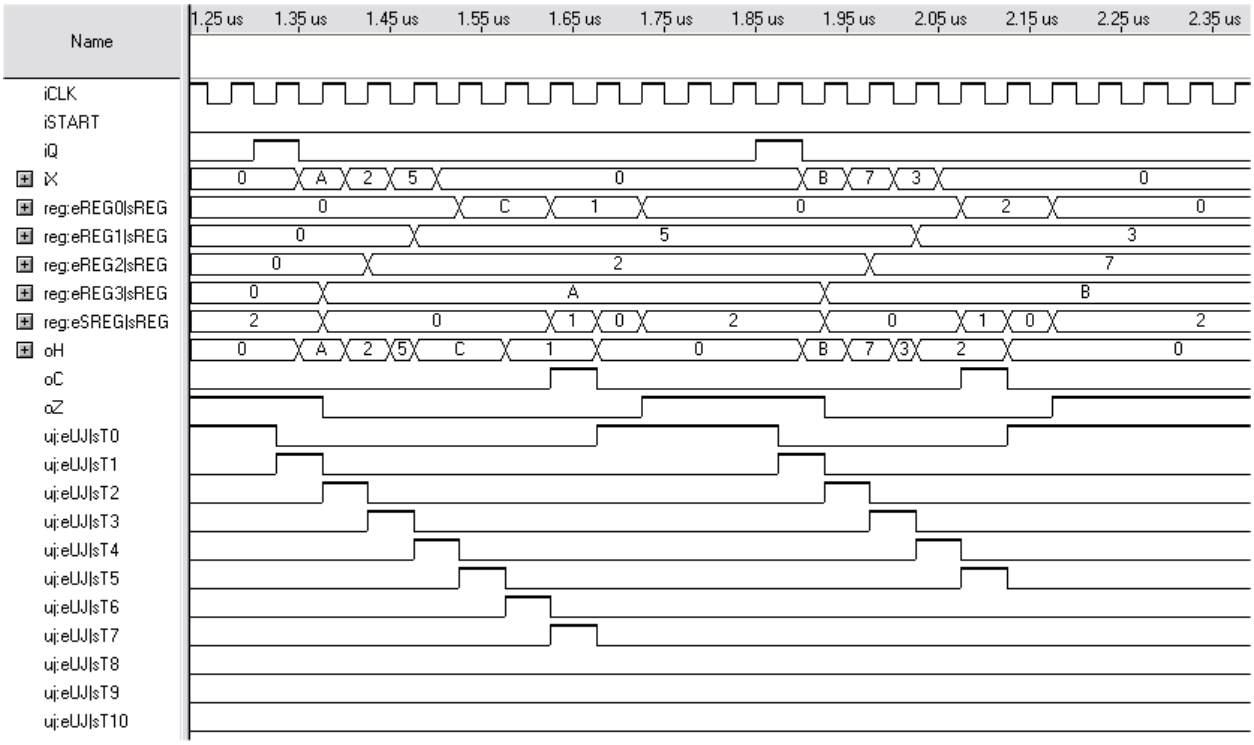
U statusnom registru je vrednost izlaznog prenosa smeštena na poziciju bita manje važnosti, dok je vrednost statusnog bita koji ukazuje da je rezultat operacije nula, smešten na poziciju bita veće važnosti. Ulazna vrednost statusnog registra je objedinjena signalom sDATA. Stanje statusnog registra je potrebno osvežavati nakon svake izvršene operacije, pa je zbog toga na ulazni signal dozvole upisa u statusni registar dovedena konstantna vrednost cONE koja je jednaka jedinici, čime se sa svakom rastućom ivicom takt signala upisuje nova vrednost u registar.

Vremenski dijagram simulacije izvršenja zadatog algoritma na realizovanom procesoru prikazuje Slika 8.80. Na slici je prikazana situacija kada su sve operacije sabiranja izvršene bez pojave statusnog bita koji ukazuje na pojavu izlaznog prenosa (carry). U prvom slučaju algoritam prolazi kroz stanja T_0 do T_9 i u registar R1 se upisuje vrednost FF_{HEX}, pošto je rezultat sabiranja (R0=R3+R2+R1=3+2+1=6) različit od nule. Drugi deo vremenskog dijagrama,

od trenutka t=700ns, prikazuje slučaj kada je rezultat sabiranja 0 i u registar `R1` se na kraju upisuje vrednost 00_{HEX}. U tom slučaju upravljačka jedinica prolazi kroz stanja T_0 do T_8 odakle prelazi u stanje T_{10}. Izvršenje algoritma u slučaju pojave izlaznog prenosa prikazuje Slika 8.81.



Slika 8.80: Simulacija izvršenja zadatog algoritma
(regularan završetak)



Slika 8.81: Simulacija izvršenja zadatog algoritma
(prevremen završetak zbog pojave carry statusnog bita)

Prvo je prikazan slučaj kada se izlazni prenos pojavio nakon druge operacije sabiranja. Ovaj izlazni prenos je detektovan proverom statusnog bita C u stanju T_7. Pošto je u tom trenutku on bio postavljen u stanje 1, upravljačka jedinica je završila izvršenje algoritma i prešla u stanje T_0. Druga situacija je prikazana od vremenskog trenutka t=1,85μs. U ovom slučaju izlazni prenos se pojavio odmah nakon izvršene prve operacije sabiranja ($B_{HEX}+7_{HEX}$). To je detektovano proverom statusnog bita C u stanju T_5, čime je upravljačka jedinica prešla u stanje T_0 i završila izvršenje algoritma.

8.19 ZADATAK:

Slika 8.82 prikazuje blok dijagram procesora koji sadži aritmetičko logičku jedinicu koja izvršava operacije koje prikazuje Tabela 8.30.

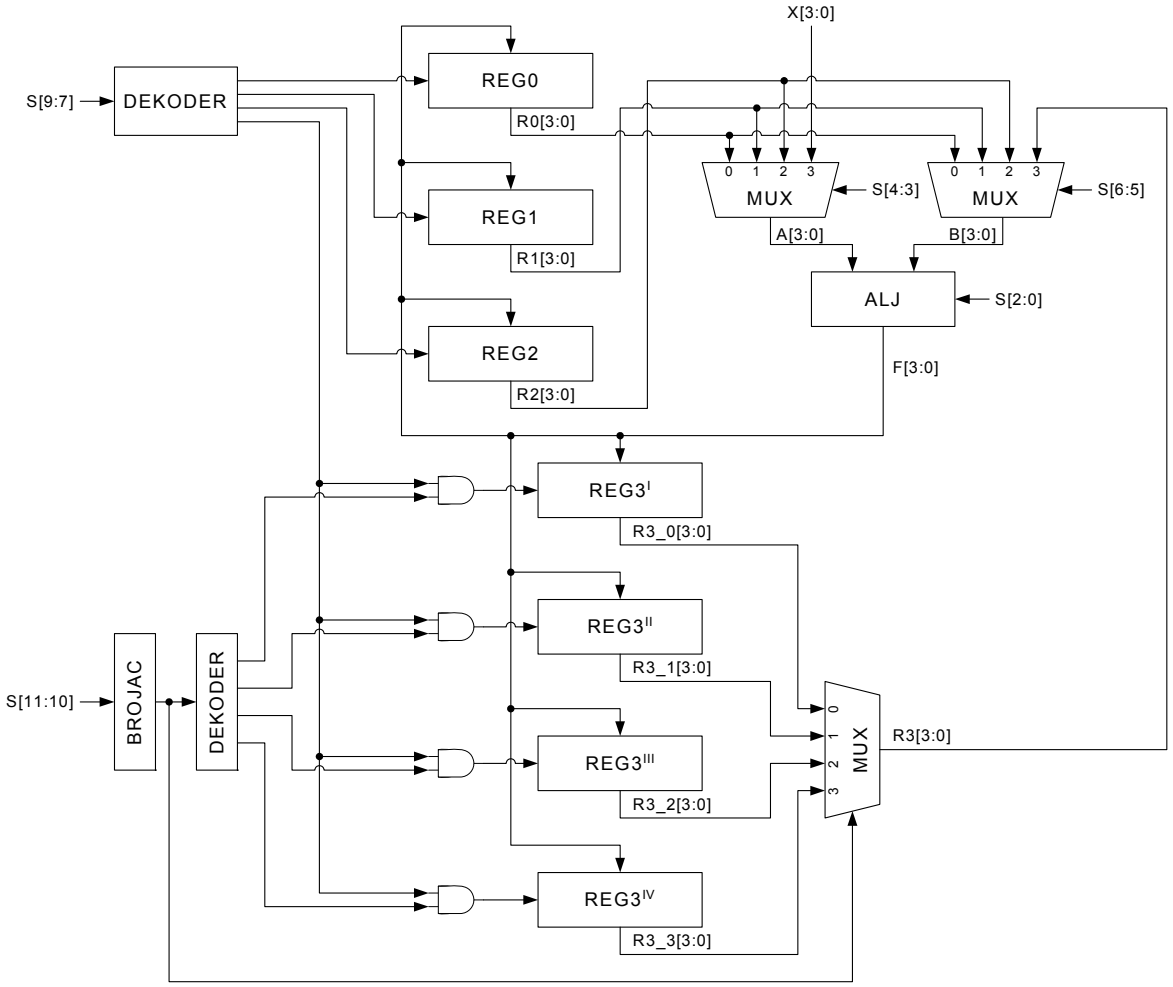
S_2	S_1	S_0	Funkcija
0	0	0	F = A
0	0	1	F = A–1
0	1	0	F = A+1
0	1	1	F = A–B
1	0	0	F = A+B
1	0	1	F = A-B-1
1	1	0	F = A+B+1
1	1	1	F = 0

Tabela 8.30: Operacije koje izvršava ALJ

Procesor ima sedam četvorobitnih registara podeljenih u dve grupe, pri čemu je prva grupa (od tri registra) dostupna direktno, dok se pristup registrima iz druge grupe (od četiri registra) obavlja preko "pokazivača" koji se nalazi u dvobitnom brojaču. Moguće je pristupiti samo onom registru koji je određen stanjem brojača. Preko ulaza INC i DEC moguće je uvećati ili umanjiti sadržaj brojača za 1 i tako pristupiti ostalim registrima iz druge grupe.

Tabela 8.31 i Tabela 8.32 opisuje značenje pojedinih bita u upravljačkoj reči procesora. Uloga upravljačkog bita S_9 je da dozvoli proces dekodovanja (S_9=1) ili ne (S_9=0). U slučaju da nije dozvoljen proces dekodovanja ne generišu se signali na izlazu dekodera, tj. nije dozvoljen upis u registre.

a) Opisati dati procesor u VHDL-u.
b) Napisati mikroprograme koji realizuju mašinske instrukcije PUSH R0 i POP R0. Instrukcija PUSH R0 smešta sadržaj registra R0 u registaj iz druge grupe, a instrukcija POP R0 sadržaj registra iz druge grupe prebacuje u registar R0. Pri radu ove dve instrukcije ažurirati stanje brojača, koje određuje registar kome se pristupa, tako da druga grupa registara ustvari predstavlja LIFO registar (stek).



Slika 8.82: Blok dijagram procesora

		S_8 S_7	S_6 S_5	S_4 S_3
0	0	R0←F	B←R0	A←R0
0	1	R1←F	B←R1	A←R1
1	0	R2←F	B←R2	A←R2
1	1	LIFO←F	B←LIFO	A←X

Tabela 8.31: Značenje pojedinih polja upravljačke reči

S_{11}	S_{10}	Q(t+1)
0	0	Q(t)
0	1	Q(t)+1
1	0	Q(t)−1
1	1	nedozvoljeno

Tabela 8.32: Značenje polja upravljačke reči za kontrolu rada brojača

REŠENJE:

Analizom blok dijagrama procesora, Slika 8.82, koji treba da se isprojektuje, vidi se da je potrebno izvršiti sintezu sledećih entiteta:

- registar,
- dvosmerni brojač,
- vektorski multiplekser 4 na 1,

- aritmetičko logička jedinica i
- dekoder 2 na 4 sa signalom dozvole dekodovanja.

Entiteti registra, multipleksera i dekodera su ekvivalentni sa modulima realizovanim u prethodnom zadatku. U odnosu na prethodni zadatak potrebno je modifikovati realizaciju aritmetičko logičke jedinice u skladu sa zadatim operacijama, Tabela 8.30, i realizovati novi modul dvosmernog brojača koji služi kao pokazivač steka.

BROJAČ:

Funkciju brojača definieše Tabela 8.32. U skladu sa zadatom tabelom, spregu ovog modula čine sledeći signali:

- `iCLK` takt signal,
- `iRESET` signal za brisanje sadržaja brojača, aktivan na visokom nivou i sinhron sa signalom takta,
- `iINC` dozvola brojanja na gore, aktivan na visokom nivou,
- `iDEC` dozvola brojanja na dole, aktivan na visokom nivou i
- `oQ` stanje brojača.

Sa ciljem formiranja generalizovanog brojača sa prizvoljnim brojem bita uvodi se generički parametar `pWIDTH` koji označava broj bita brojača koji se instancira. Pretpostavlja se da instancirani brojač ima četiri bita.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;

ENTITY BROJAC IS
  GENERIC (
    pWIDTH: integer := 4    -- pretpostavljeni broj bita je 4
  );
  PORT(
    iCLK:     IN  STD_LOGIC;
    iRESET:   IN  STD_LOGIC;
    iINC:     IN  STD_LOGIC;
    iDEC:     IN  STD_LOGIC;
    oQ:       OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
  );
END BROJAC;

ARCHITECTURE ARH_BROJAC OF BROJAC IS
  -- stanje brojaca
  SIGNAL sCNT: UNSIGNED(pWIDTH-1 DOWNTO 0);
  -- vektor za upravljanje stanjem brojaca
  SIGNAL sCTRL: STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN
  -- formiranje vektora za upravljanje sa brojacem
  sCTRL <= (iDEC & iINC);
```

```
    PROCESS(iCLK) BEGIN
      IF (iCLK'EVENT AND iCLK='1') THEN
        IF (iRESET = '1') THEN -- sinhroni reset
          sCNT <= (OTHERS => '0');
        ELSE
          CASE sCTRL IS
            WHEN "00"   => sCNT <= sCNT;       -- zadrzavanje stanja
            WHEN "01"   => sCNT <= sCNT + 1;   -- brojanje na gore
            WHEN "10"   => sCNT <= sCNT - 1;   -- brojanje na dole
            WHEN OTHERS => sCNT <= sCNT;       -- zadrzavanje stanja
          END CASE;
        END IF;
      END IF;
    END PROCESS;

    -- preslikavanje stanja brojaca na izlazni vektor
    oQ <= STD_LOGIC_VECTOR(sCNT);

END ARH_BROJAC;
```

Radi lakše VHDL realizacije kontrole brojanja brojača uveden je signal `sCTRL` koji objedinjuje ulazne signale dozvole brojanja na gore (`iINC`) i na dole (`iDEC`). Pomoću ovog signala se jednostavno realizuje kontrola brojanja uslovnim `CASE` iskazom koji je u skladu sa zadatom funkcionalnošću brojača, Tabela 8.32.

ALJ:

Aritmetičko logička jedinica služi za izvršenje zadatih operacija (Tabela 8.30). Sprežni signali tražene ALJ su sledeći:

- `iA` prvi ulazni četvorobitni operand,
- `iB` drugi ulazni četvorobitni operand,
- `iSEL` adresni ulaz ALJ za odabir operacije,
- `oH` rezultat izvršene aritmetičko logičke operacije i
- `oC` izlazni prenos.

ALJ je sintetizovana pomoću vektora na način koji je ranije prikazan u poglavlju posvećenom projektovanju Aritmetičko logičkih jedinica.

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE ieee.numeric_std.all;

ENTITY ALJ IS PORT(
  iA, iB: IN  std_logic_vector(3 DOWNTO 0);
  iSEL:   IN  std_logic_vector(2 DOWNTO 0);
  oH:     OUT std_logic_vector(3 DOWNTO 0);
  oC:     OUT std_logic );
END ALJ;
```

```
ARCHITECTURE ARH_ALJ OF ALJ IS
  -- operandi koji su prosireni za jedan bit
  -- zbog racunanja izlaznog prenosa
  SIGNAL sA, sB: unsigned(4 DOWNTO 0);
  -- rezultat operacije, ulaz u pomerac
  -- prosiren za jedan bit gde ce se smestiti
  -- izracunati izlazni prenos
  SIGNAL sF:  unsigned(4 DOWNTO 0);
BEGIN
  sA   <= unsigned('0' & iA);  -- operand A prosiren za jedan bit
  sB   <= unsigned('0' & iB);  -- operand B prosiren za jedan bit

  PROCESS(iSEL, sA, sB) BEGIN
    CASE iSEL IS
      WHEN "000"  => sF <= sA;
      WHEN "001"  => sF <= sA - 1;
      WHEN "010"  => sF <= sA + 1;
      WHEN "011"  => sF <= sA - sB;
      WHEN "100"  => sF <= sA + sB;
      WHEN "101"  => sF <= sA - sB - 1;
      WHEN "110"  => sF <= sA + sB + 1;
      WHEN OTHERS => sF <= "00000";
    END CASE;
  END PROCESS;

  -- formiranje izlaza ALJ
  oH <= std_logic_vector(sF(3 DOWNTO 0));
  oC <= std_logic(sF(4));

END ARH_ALJ;
```

PROCESOR:

U ovom modulu se instanciraju svi prethodno opisani moduli i međusobno povezuju u skladu sa blok dijagramom procesora, Slika 8.82.

Prvo je potrebno formirati pakovanje sa opisom sprega svih modula koji se instanciraju:

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;

PACKAGE CPU_PKG IS
  COMPONENT REG
    GENERIC (
      pWIDTH: integer := 4
    );
    PORT (
      iCLK, iCLR: IN  STD_LOGIC;
      iCE: IN  STD_LOGIC;
      iD:  IN  STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0);
      oQ:  OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
    );
  END COMPONENT;
```

```
  COMPONENT MUX4x1 IS
    GENERIC (
      pWIDTH: integer := 4
    );
    PORT (
      iX0, iX1,
      iX2, iX3: IN  std_logic_vector(pWIDTH-1 DOWNTO 0);
      iSEL:     IN  std_logic_vector(1 DOWNTO 0);
      oY:       OUT std_logic_vector(pWIDTH-1 DOWNTO 0)
    );
  END COMPONENT;

  COMPONENT DECODER IS
    PORT(
      iX:  IN  std_logic_vector (1 DOWNTO 0);
      iEN: IN  std_logic;
      oY:  OUT std_logic_vector (3 DOWNTO 0)
    );
  END COMPONENT;

  COMPONENT ALJ IS
    PORT(
      iA, iB: IN  std_logic_vector(3 DOWNTO 0);
      iSEL:   IN  std_logic_vector(2 DOWNTO 0);
      oH:     OUT std_logic_vector(3 DOWNTO 0);
      oC:     OUT std_logic
    );
  END COMPONENT;

  COMPONENT BROJAC
    GENERIC (
      pWIDTH: integer := 4
    );
    PORT(
      iCLK:   IN  STD_LOGIC;
      iRESET: IN  STD_LOGIC;
      iINC:   IN  STD_LOGIC;
      iDEC:   IN  STD_LOGIC;
      oQ:     OUT STD_LOGIC_VECTOR(pWIDTH-1 DOWNTO 0)
    );
  END COMPONENT;

END CPU_PKG;
```

Nakon formiranja pakovanja sa opisom sprega svih komponenti koje se instanciraju prelazi se na opis entiteta projektovanog procesora. Projektovani modul procesora sadrži sledeće ulazno/izlazne signale:

- `iCLK` takt signal,
- `iRESET` signal za brisanje sadržaja svih registara i brojača, aktivan na visokom nivou i sinhron sa signalom takta,
- `iX` četvorobitni ulazni vektor,
- `iSEL` upravljačka reč procesora,

- oALJ_SEL deo upravljačke reči procesora rezervisan za odabir aritmetičko logičke operacije,
- oAMUX_SEL deo upravljačke reči procesora rezervisan za odabir operanda A,
- oBMUX_SEL deo upravljačke reči procesora rezervisan za odabir operanda B,
- oREG_SEL deo upravljačke reči procesora rezervisan za odabir registra u koji će se smestiti rezultat aritmetičko logičke operacije,
- oREG_WE deo upravljačke reči procesora rezervisan za generisanje signala dozvole upisa u registre,
- oINC deo upravljačke reči procesora rezervisan za generisanje signala dozvole brojanja brojača na gore,
- oDEC deo upravljačke reči procesora rezervisan za generisanje signala dozvole brojanja brojača na dole,
- oH rezultat izvršene aritmetičko logičke operacije, četvorobitni izlazni vektor,
- oC stanje statusnog bita koji ukazuje na izlazni prenos prethodno izvršene operacije.

Ulazna upravljačka reč procesora, iSEL, je u okviru arhitekture procesora podeljena na delove za upravljanje sa pojedinim modulima procesora u skladu sa zadatkom. Podela je izvršena uvođenjem aliasa na odgovarajuće delove upravljačke reči, npr:

```
ALIAS sAMUX_SEL: STD_LOGIC_VECTOR(1 DOWNTO 0) IS iSEL(4 DOWNTO 3);
```

Na ovaj način se u okviru arhitekture lakše razaznaje koja je uloga svih nezavisnih delova upravljačke reči.

Sa ciljem lakše analize rada isprojektovanog procesora svi aliasi su prosleđeni na izlazne pinove radi efikasnijeg praćenja vremenskih dijagrama rada procesora.

VHDL opis procesora je sledeći:

```
LIBRARY ieee;
  USE ieee.std_logic_1164.all;
  USE work.CPU_PKG.all; -- ukljucivanje pakovanja sa komponentama

ENTITY CPU IS
  PORT (
    iCLK:      IN  STD_LOGIC;
    iRESET:    IN  STD_LOGIC;
    iX:        IN  STD_LOGIC_VECTOR(3 DOWNTO 0);
    iSEL:      IN  STD_LOGIC_VECTOR(11 DOWNTO 0);
    oALJ_SEL:  OUT STD_LOGIC_VECTOR(2 DOWNTO 0);
    oAMUX_SEL: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
    oBMUX_SEL: OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
```

```
    oREG_SEL:  OUT STD_LOGIC_VECTOR(1 DOWNTO 0);
    oREG_WE:   OUT STD_LOGIC;
    oINC:      OUT STD_LOGIC;
    oDEC:      OUT STD_LOGIC;
    oH:        OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
    oC:        OUT STD_LOGIC
  );
END CPU;

ARCHITECTURE ARH_CPU OF CPU IS
  -- konstanta 1
  CONSTANT cONE: STD_LOGIC := '1';
  -- formiranje aliasa na ulazni upravljacki vektor
  -- radi lakse kontrole pojedinih polja upravljacke reci
  ALIAS sALJ_SEL:  STD_LOGIC_VECTOR(2 DOWNTO 0) IS iSEL(2 DOWNTO 0);
  ALIAS sAMUX_SEL: STD_LOGIC_VECTOR(1 DOWNTO 0) IS iSEL(4 DOWNTO 3);
  ALIAS sBMUX_SEL: STD_LOGIC_VECTOR(1 DOWNTO 0) IS iSEL(6 DOWNTO 5);
  ALIAS sREG_SEL:  STD_LOGIC_VECTOR(1 DOWNTO 0) IS iSEL(8 DOWNTO 7);
  ALIAS sREG_WE:   STD_LOGIC IS iSEL(9);
  ALIAS sINC:      STD_LOGIC IS iSEL(10);
  ALIAS sDEC:      STD_LOGIC IS iSEL(11);
  -- registri
  SIGNAL sR0, sR1, sR2, sR3: STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL sR30, sR31, sR32, sR33: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- ulazni vektori u ALJ
  SIGNAL sA, sB: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- rezultat ALJ
  SIGNAL sF: STD_LOGIC_VECTOR(3 DOWNTO 0);
  SIGNAL sC: STD_LOGIC;  -- izlazni prenos
  -- izlaz dekodera -> dozvola upisa u registre
  SIGNAL sWE: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- izlaz dekodera -> dozvola upisa u stek registre
  SIGNAL sREG3_EN, sWE3: STD_LOGIC_VECTOR(3 DOWNTO 0);
  -- vrednost brojaca
  SIGNAL sCNT: STD_LOGIC_VECTOR(1 DOWNTO 0);
BEGIN

  -- dozvola upisa u registre
  eDEKODER: DECODER
    PORT MAP(iX=>sREG_SEL, iEN=>sREG_WE, oY=>sWE);

  -- registar na adresi 0
  eREG0: REG
    GENERIC MAP (pWIDTH=>4)
    PORT    MAP (iCLK=>iCLK, iCLR=>iRESET, iCE=>sWE(0),
                 iD=>sF, oQ=>sR0);
  -- registar na adresi 1
  eREG1: REG
    GENERIC MAP (pWIDTH=>4)
    PORT    MAP (iCLK=>iCLK, iCLR=>iRESET, iCE=>sWE(1),
                 iD=>sF, oQ=>sR1);

  -- registar na adresi 2
  eREG2: REG
    GENERIC MAP (pWIDTH=>4)
    PORT    MAP (iCLK=>iCLK, iCLR=>iRESET, iCE=>sWE(2),
                 iD=>sF, oQ=>sR2);
```

```
-- brojac -> ukazivac steka
eCNT: BROJAC
  GENERIC MAP (pWIDTH=>2)
  PORT    MAP (iCLK=>iCLK, iRESET=>iRESET,
               iINC=>sINC, iDEC=>sDEC, oQ=>sCNT);

-- dozvola upisa u stek registre
eSTACK_DEC: DECODER
  PORT MAP(iX=>sCNT, iEN=>cONE, oY=>sREG3_EN);

-- odredjivanje signala dozvole upisa u stek registre
sWE3(0) <= sREG3_EN(0) AND sWE(3);
sWE3(1) <= sREG3_EN(1) AND sWE(3);
sWE3(2) <= sREG3_EN(2) AND sWE(3);
sWE3(3) <= sREG3_EN(3) AND sWE(3);

-- stek registri na adresi 3
eREG30: REG
  GENERIC MAP (pWIDTH=>4)
  PORT    MAP (iCLK=>iCLK, iCLR=>iRESET, iCE=>sWE3(0),
               iD=>sF, oQ=>sR30);
eREG31: REG
  GENERIC MAP (pWIDTH=>4)
  PORT    MAP (iCLK=>iCLK, iCLR=>iRESET, iCE=>sWE3(1),
               iD=>sF, oQ=>sR31);
eREG32: REG
  GENERIC MAP (pWIDTH=>4)
  PORT    MAP (iCLK=>iCLK, iCLR=>iRESET, iCE=>sWE3(2),
               iD=>sF, oQ=>sR32);
eREG33: REG
  GENERIC MAP (pWIDTH=>4)
  PORT    MAP (iCLK=>iCLK, iCLR=>iRESET, iCE=>sWE3(3),
               iD=>sF, oQ=>sR33);

-- multiplekser za odabir izlaznog registra iz steka
eMUXR3: MUX4x1
  PORT MAP(iX0=>sR30, iX1=>sR31, iX2=>sR32, iX3=>sR33,
           iSEL=>sCNT, oY=>sR3);

-- multiplekser za odabir operanda A
eMUXA: MUX4x1
  PORT MAP(iX0=>sR0, iX1=>sR1, iX2=>sR2, iX3=>iX,
           iSEL=>sAMUX_SEL, oY=>sA);

-- multiplekser za odabir operanda B
eMUXB: MUX4x1
  PORT MAP(iX0=>sR0, iX1=>sR1, iX2=>sR2, iX3=>sR3,
           iSEL=>sBMUX_SEL, oY=>sB);

-- ALJ
eALJ: ALJ
  PORT MAP(iA=>sA, iB=>sB, iSEL=>sALJ_SEL, oH=>sF, oC=>sC);

-- izlazni signali
oALJ_SEL  <= sALJ_SEL;
oAMUX_SEL <= sAMUX_SEL;
oBMUX_SEL <= sBMUX_SEL;
oREG_SEL  <= sREG_SEL;
```

```
    oREG_WE   <= sREG_WE;
    oINC      <= sINC;
    oDEC      <= sDEC;
    oH        <= sF;
    oC        <= sC;
END ARH_CPU;
```

Instrukcija `PUSH R0` se može realizovati u jednom taktu tako što se sadržaj registra R0 dovede na A ulaz aritmetičko logičke jedinice, izabere se operacija `F=A` i rezultat F se upiše u grupu registara R3, odnosno u stek memoriju. Istovremeno se uvećava vrednost brojača čime se vrednost brojača postavlja na prvu sledeću slobodnu lokaciju u stek memoriji. Upravljačka reč procesora za izvršenje ove operacije ima sledeći oblik:

DEC	INC	WE	REG_SEL			BMUX_SEL		AMUX_SEL		ALJ_SEL	
S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
0	1	1	1	1	0	0	0	0	0	0	0

Zbog izabrane operacije, `F=A`, nije bitna vrednost upravljačkih signala za odabir operanda B aritmetičko logičke jedinice. U ovom slučaju odabrana je vrednost `BMUX_SEL=00`.

Instrukcija `POP R0` se mora realizovati tokom dve periode takta. To je iz razloga što se izlaz R3 grupe registara može dovesti samo na B ulaz aritmetičko logičke jedinice, a ne postoji operacija za propuštanje operanda B. Da bi se to realizovalo može se iskoristiti operacija `F=A+B`, gde će se obezbediti da je vrednost operanda A jednaka nuli. Tako će prva mikroinstrukcija predstavljati upis vrednosti 0 u registar R0, dok će druga mikroinstrukcija izvršiti transfer podataka sa vrha R3 grupe registara i umanjiti sadržaj brojača za jedan. Upravljačka reč procesora za izvršenje ove operacije ima sledeći oblik:

DEC	INC	WE	REG_SEL			BMUX_SEL		AMUX_SEL		ALJ_SEL	
S_{11}	S_{10}	S_9	S_8	S_7	S_6	S_5	S_4	S_3	S_2	S_1	S_0
0	0	1	0	0	0	0	0	0	1	1	1
1	0	1	0	0	1	1	0	0	1	0	0

Tokom postavljanja registra R0 na vrednost nula, prva mikroinstrukcija, odabrana je operacija aritmetičko logičke jedinice `F=0`. Stoga nije bitno koji se operandi dovode na ulaze aritmetičko logičke jedinice pošto oni ne učestvuju u izvršenju operacije. U ovom slučaju postavljene su vrednosti `AMUX_SEL=00` i `BMUX_SEL=00`.

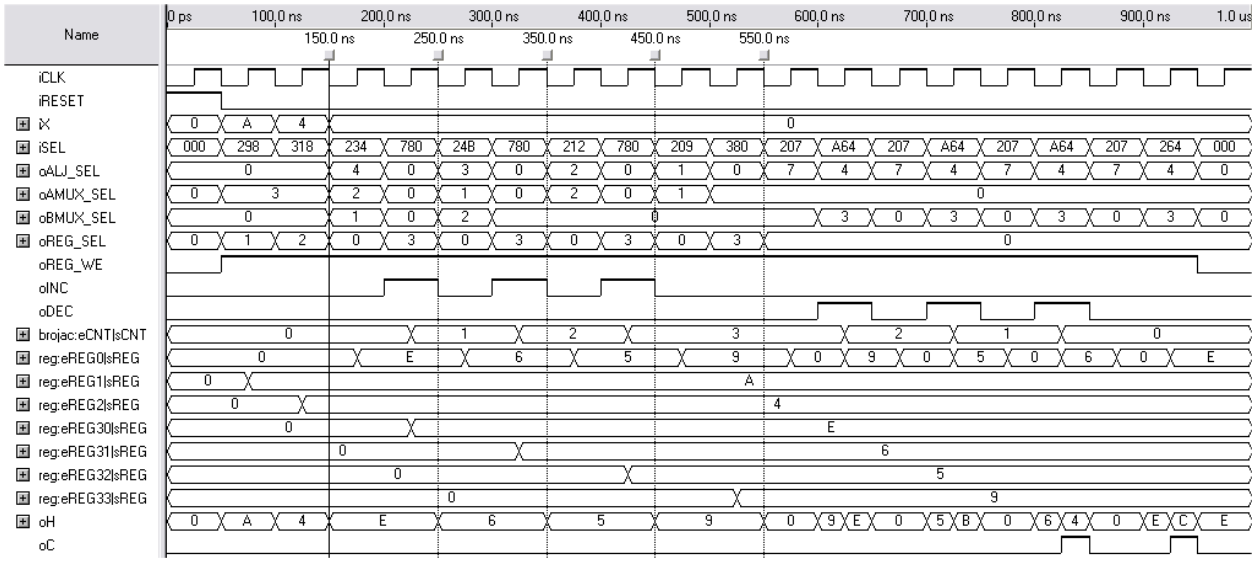
Radi ilustracije rada realizovanog procesora, kao i mikroinstrukcija `PUSH R0` i `POP R0` navodi se vremenski dijagram, Slika 8.83, sa prikazom izvršenja sledećeg mikroprograma:

```
R1 ← X
R2 ← X
R0 ← R1 + R2
PUSH R0
R0 ← R1 - R2
PUSH R0
R0 ← R2 + 1
PUSH R0
R0 ← R1 - 1
PUSH R0
POP R0
POP R0
POP R0
POP R0
```

Na početku vremenskog dijagrama se prvo u registar R1 upisuje vrednost A_{HEX}, i odmah zatim u registar R2 se upisuje vrednost 4_{HEX}.

U vremenskom intervalu od 150ns do 250ns se izvršavaju mikroinstrukcije:

```
R0 ← R1 + R2
PUSH R0
```



Slika 8.83: Ilustracija rada realizovanog procesora

Rezultat prve mikroinstrukcije (R1+R2=E_{HEX}) se smešta u prvi registar grupe registar R3, tj. u registar R30. Vrednost brojača `sCNT` je uvećana za jedan (`sINC=1`). Sledi izvršenje druge dve mikroinstrukcije, u vremenskom imtervalu od 250ns do 350ns, gde se razlika R1-R2 smešta u grupu registara R3. Time se razlika R1-R2=6_{HEX}, smešta u registar R31 i uvećava se vrednost brojača `sCNT` za jedan. U sledećem označenom vremenskom intervalu, od 350ns do 450ns, u LIFO

organizaciju registara se upisuje inkrementirana vrednost registra R2 (R2+1). Zbog toga registar R32 prima vrednost 5_{HEX} i vrednost brojača `sCNT` se još jednom uvećava za jedan. Tokom vremenskog intervala od 450ns do 550ns se u grupu registara R3 upisuje dekrementirana vrednost registra R1 (R1-1). Rezultat ove mikroinstrukcije, R1-1=9_{HEX}, se upisuje u registar R33. Prilikom izvršenja ove, četvrte po redu, `PUSH R0` mikroinstrukcije nije uvećana vrednost brojača. To nije urađeno iz razloga što bi se time vrednost brojača promenila na 0, pošto je dvobitni brojač prethodno dostigao svoju maksimalnu vrednost. Na ovaj način, bez uvećanja vrednosti brojača, se sa sledećom instrukcijom (`POP R0`) očitava poslednja upisana vrednost u R3 grupu registara čime se simulira LIFO organizacija registara.

Sledi izvršenje četiri mikroinstrukcije `POP R0`. Sa vremenskog dijagrama se vidi da je za svaku mikroinstrukciju potrebno dve periode takt signala, gde se u prvoj periodi u registar R0 upisuje vrednost 0, dok se u drugoj periodi izvršava mikroinstrukcija R0←R0+R3. Sa izvršenjem svake mikroinstrukcije `POP R0` smanjuje se vrednost brojača `sCNT` za jedan aktiviranjem signala `sDEC`. Kao i kod popunjavanja LIFO organizacije registara, i ovde se prilikom očitavanja poslednje upisane vrednosti u registre R3 ne menja vrednost brojača radi očuvanja vrednosti pokazivača steka na prvu lokaciju u koju se upisuje novi sadržaj instrukcijom `PUSH R0`.