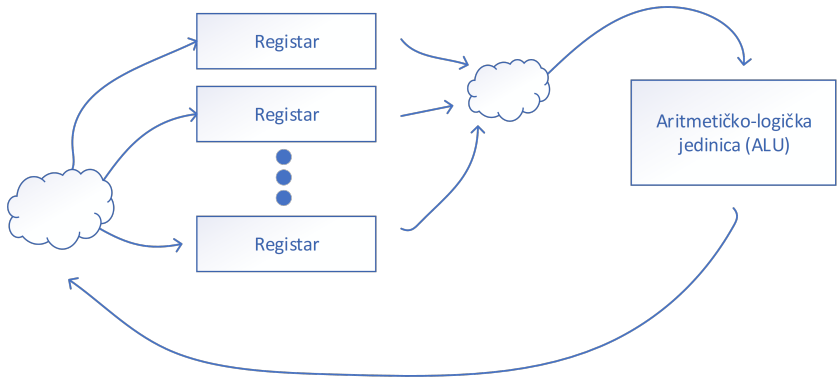
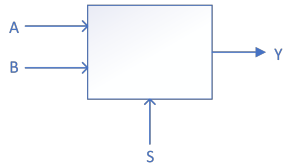
A2. Auditorna vežba 2: Kombinacione mreže



Prvi detalj: izbor operanda

Podsetimo se skice osnovne putanje izvršavanja instrukcije u procesoru koju smo prošli put predstavili. U desnom "oblačiću" treba da rešimo **problem izbora operanada** koji ulaze u željenu operaciju, prilikom izvršavanja jedne instrukcije. Od ukupno M registara koje imamo na raspolaganju, ta komponenta treba da prosledi svega jedan ili dva, zavisno od operacije. Na primer, ako je u pitanju operacija sabiranja, čiji smo sistem projektovali prošli put, potrebna su nam dva operanda.

Elementarni problem koji ćemo rešiti je: **izbor jedne od dve vrednosti**. Odn. treba da osmislimo komponentu koja će moći da prosledi jednu od dve vrednosti, u zavisnosti od signala na kome piše koju vrednost treba proslediti. Posle toga će biti jednostavno proširiti ovo rešenje na problem izbora N od M vrednosti.



Komponenta koja rešava ovaj elementarni problem ima ulaze i izlaze predstavljene na gornjoj slici. Izlaz Y je jednak A ukoliko je S na jednoj vrednosti, odn. jednak je ulazu B ukoliko je ulaz S na drugoj vrednosti.

$$Y = \begin{cases} A, & S = 0 \\ B, & S = 1 \end{cases}$$

Projektovanje komponente će biti isto kao i u slučaju sabirača prošli put:

- potrebno je da definišemo **ponašanje željene komponente**, što možemo učiniti opisom funkcionalnosti kao u prethodnoj jednačini ili pomoću istinitosne tablice;
- iz željenog ponašanja, izvodimo **Bulovu jednačinu** za svaki izlaz komponente;
- proverimo da li je data jednačina minimalna ili možemo da je optimizujemo postupkom **minimizacije**;
- na osnovu izvedene jednačine, projektujemo sistem crtajući **logičku šemu** sistema.

Istinitosna tablica željene komponente je sledeća:

S	A	B	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Jednačinu možemo izvesti iz tablice ili iz logike ponašanja koju smo iznad definisali matematički.

$$Y = A * \bar{S} + B * S$$

Iz jednačine, možemo nacrtati šemu i simulirati je u nekom od alata za simulaciju digitalnih sistema.

Kako uopštavamo ovu komponentu na izbor N vrednosti od M ulaznih vrednosti?

- Svaki izlaz se može posmatrati kao posebna komponenta koja bira 1 od M ulaznih vrednosti, dakle konačno rešenje ima N istih komponenti.
- Proširenje podrške sa 2 na M ulaza se radi na sledeći način:
  - Proširujemo broj selekcionih bita (S) na potreban broj tako da pokrijemo svih M mogućnosti. Potreban broj bita je $\lceil \log_2 M \rceil$ (najbliži ceo broj veći ili jednak logaritmu po osnovi 2 od ukupnog broja mogućnosti).
  - Proširujemo broj ulaza na M.
  - Formiramo veću tablicu i dužu jednačinu na isti način kao i u elementarnom slučaju.

Komponenta koju smo definisali se zove **multiplekser**.

Drugi detalj: izbor registra rezultata

Zadatak levog "oblačića" na gornjoj šemi je da rezultat iz aritmetičko-logičke jedinice upiše u **tačno jedan registar** koji želimo da bude registar rezultata u instrukciji koja se trenutno izvršava u procesoru.

! Ovaj problem ćemo rešiti tako što ćemo **rezultat proslediti svakom registru**, međutim uz rezultat ćemo registru definisati i **signal dozvole preuzimanja rezultata**. Registar će vrednost preuzeti, odn. svoju vrednost promeniti, samo u slučaju da je aktiviran signal dozvole; u suprotnom, registar neće promeniti svoju vrednost.

Sada ćemo projektovati komponentu koja **definiše signal dozvole** tačno jednom registru, onom čiji redni broj se nalazi na ulazu komponente.

- Ulaz (X): redni broj registra kome šaljemo signal dozvole.
- Izlaz (Y): signali dozvole koji se šalju svim registrima u sistemu.

⚠ Kao primer, posmatraćemo slučaj u kome imamo 4 registra. Tada nam treba 2 bita za kodovanje svih rednih brojeva koji mogu da se nađu na ulazu (0, 1, 2, 3).

Istinitosna tablica željene komponente je sledeća:

X1	X0	Y3	Y2	Y1	Y0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Jednačine izlaza:

$$Y_3 = X_1 * X_0$$

$$Y_2 = X_1 * \overline{X_0}$$

$$Y_1 = \overline{X_1} * X_0$$

$$Y_0 = \overline{X_1} * \overline{X_0}$$

U opštem slučaju, ova komponenta će imati N ulaza i 2^N izlaza.

★ Komponenta koju smo definisali se zove **dekoder**.

Kombinaciona mreža

★ U opštem slučaju, **kombinaciona mreža** je digitalni sistem koji računa Bulovu funkciju, odn. digitalni sistem kod koga je **izlaz u svakom trenutku Bulova funkcija ulaza**. Vrednost izlaza zavisi **samo od trenutnih vrednosti** ulaza, a ne i od vrednosti koje su ranije bile na tim ulazima.

★ Važne osobine kombinacionih mreža su sledeće:

1. Funkcija koja opisuje svaki izlaz kombinacione mreže mora biti **potpuno definisana**.
2. Funkcija koja opisuje svaki izlaz kombinacione mreže mora biti **jednoznačna**.
3. **Ulaz** kombinacione mreže **ne sme biti funkcija izlaza** iste kombinacione mreže, direktno ili indirektno. Tj. ne sme biti kombinacione povratne sprege.

Osobina 1. Potpuna definisanost sledi iz matematičke osobine funkcije da je definisana u svom domenu. Da bi naša kombinaciona mreža bila potpuno definisana, odn. deterministička, izlaz mora biti definisan za svaku kombinaciju koja se može naći na ulazu. U suštini, to znači da istinitosna tablica mora biti potpuno popunjena.

Osobina 2. Jednoznačnost takođe sledi iz matematičke osobine funkcije da za datu vrednost ulazne promenljive ima tačno jednu vrednost izlazne promenljive.
- U našem slučaju to znači da, matematički, svaki red u istinitosnoj tablici ima tačno jedno vrednost u svakoj izlaznoj koloni.
- Sa stanovišta sistema to znači da svaki signal može da ima **samo jedan izvor vrednosti**.

Osobina 3. Povratna sprega, tzv. kombinaciona petlja, nastaje kada izlaz direktno spojimo na ulaz iste komponente ili indirektno, kroz druge kombinacione mreže. Ovo bi matematički bila rekurzivna definicija funkcije, što nije nemoguće. Međutim, ovakav sistem električno ne može da funkcioniše kao logička funkcija, onako kako smo ih definisali u tehnologijama logičkih kola. Naprotiv, sistem koji se tada dobije počinje da zavisi od svojih prethodnih vrednosti i ne spada u kombinacione mreže.

Standardne kombinacione mreže

U nastavku sledi "*cheat sheet*" (tzv. puškica) koja izlistava najvažnije standardne kombinacione mreže koje mogu da se pojave kao komponente složenijih kombinacionih mreža. Za svaku standardnu kombinacionu mrežu su dati: opis željene funkcionalnosti, jednačine za izlaze (za primer određene veličine) i uobičajen način, ali ne i jedini, za opis te komponente u VHDL jeziku za opis digitalnih sistema.

Multiplekser

★ **Funkcija**: izlaz je jednak jednom od ulaza u zavisnosti od vrednosti ulaza selekcije.

Ulazi: N signala (od proizvoljno bita) od kojih se na izlaz prosleđuje samo jedan; jedan signal selekcije ($\log_2 N$ bita).
Izlazi: jedan signal (isto bita kao i svaki ulaz).

⚠ **Primer:** multiplekser koji bira jedan od 4 ulaza, u zavisnosti od 2-bitne selekcije. Dat je i primer VHDL opisa multipleksera.

$iSEL_1$	$iSEL_0$	oY
0	0	iX0
0	1	iX1
1	0	iX2
1	1	iX3

```
oY <= iX0 when iSEL = "00" else
      iX1 when iSEL = "01" else
      iX2 when iSEL = "10" else
      iX3;
```

Dekoder

Funkcija: tačno jedan bit na izlazu se postavlja na 1, i to onaj čiji se redni broj (indeks) nalazi na ulazu kao binarni broj.

Ulazi: N-bitni ulaz iX.
Izlazi: ukupno 2^N bita izlaza oY.

Primer: dekoder koji šalje jedinicu na jedan od 4 bita izlaza (dakle, ima 2 bita na ulazu).

iX_1	iX_0	oY_3	oY_2	oY_1	oY_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

```
oY <= "0001" when iX = "00" else
      "0010" when iX = "01" else
      "0100" when iX = "10" else
      "1000";
```

Demultiplekser

Funkcija: jedan ulaz se prosleđuje na tačno jedan od izlaza, i to na onaj čiji redni broj (indeks) se nalazi na selekcionom ulazu; na ostale izlaze se postavlja 0.

Ulazi: jedan ulaz iX (proizvoljno bita), jedan signal selekcije ($\log_2 N$ bita).
Izlazi: N izlaza takvih da svaki ima po onoliko bita koliko i ulaz.

Primer: demultiplekser koji prosleđuje ulaz na jedan od 4 izlaza, u zavisnosti od 2-bitne selekcije.

$iSEL_1$	$iSEL_0$	$oY3$	$oY2$	$oY1$	$oY0$
0	0	0	0	0	iX
0	1	0	0	iX	0
1	0	0	iX	0	0
1	1	iX	0	0	0

```
oY3 <= iX when iSEL = "00" else (others => '0');
oY2 <= iX when iSEL = "01" else (others => '0');
oY1 <= iX when iSEL = "10" else (others => '0');
oY0 <= iX when iSEL = "11" else (others => '0');
```

Koder

Funkcija: na izlazu se pojavljuje redni broj (indeks) bita ulaza koji je na vrednosti 1; ukoliko nijedan bit ulaza nije na vrednosti 1, to se pokazuje na posebnom izlazu.
Modifikacija ponašanja (prioritetni koder): ukoliko se više bita nalazi na vrednosti 1, na izlazu je redni broj najprioritetnijeg ulaza.

Ulazi: ukupno 2^N bita ulaza iX.
Izlazi: ukupno N bita izlaza oY; poseban bit izlaza (oVALID) koji pokazuje da li je bar jedan bit na ulazu na vrednosti 1 ili nije.

Primer: prioritetni koder sa 4 ulaza i 2 izlaza, takav da su biti sa višim indeksom većeg prioriteta.

iX_3	iX_2	iX_1	iX_0	oY_1	oY_0	$oVALID$
1	-	-	-	1	1	1
0	1	-	-	1	0	1
0	0	1	-	0	1	1
0	0	0	1	0	0	1
0	0	0	0	-	-	0

```
oY <= "11" when iX(3) = '1' else
      "10" when iX(2) = '1' else
      "01" when iX(1) = '1' else
      "00" when iX(0) = '1' else
      "--";

oVALID <= '0' when iX = "0000" else '1';
```

Komparator

Funkcija: izlaz je na vrednosti 1 ako je zadovoljena relacija koju želimo da proverimo komparatorom, u suprotnom je 0.

Ulazi: dva signala (proizvoljno bita) koja poredimo željenom relacijom.
Izlazi: jedan bit koji pokazuje zadovoljenje željene relacije.

Primer: komparator koji poredi jednakost dva 8-bitna broja.

$$\text{oY} = \begin{cases} 1, & iA = iB \\ 0, & iA \neq iB \end{cases}$$

```
oY <= '1' when iA = iB else '0';
```

Još neki primeri u VHDL-u:

komparator koji proverava različitost dva signala	`oY <= '1' when iA /= iB else '0';`
komparator koji proverava relaciju "manje od"	`oY <= '1' when iA < iB else '0';`
komparator koji proverava relaciju "manje od ili jednako"	`oY <= '1' when iA <= iB else '0';`
komparator koji proverava relaciju "veće od"	`oY <= '1' when iA > iB else '0';`

komparator koji proverava relaciju "veće od ili jednako"	`oY <= '1' when iA >= iB else '0';`
komparator koji proverava jednakost sa konstantom	`oY <= '1' when iA = 0 else '0';`

Pomerač

Funkcija: izlaz je pomerena vrednost ulaza za željeni broj mesta koji može biti drugi ulaz komponente.
Varijante:

- Logičko (ili aritmetičko) pomeranje u levo: pomeramo sve bite u levo za željeni broj mesta i na prazna mesta ubacujemo nule.
- Logičko pomeranje u desno: pomeramo sve bite u desno za željeni broj mesta i na prazna mesta ubacujemo nule.
- Aritmetičko pomeranje u desno: pomeramo sve bite u desno za željeni broj mesta i na prazna mesta ubacujemo bit znaka ulaznog broja.

Logičko pomeranje predstavlja množenje (u levo) ili deljenje (u desno) sa 2 **za neoznačene brojeve.**
Aritmetičko pomeranje predstavlja množenje (u levo) ili deljenje (u desno) sa 2 **za označene brojeve**.

Ulazi: jedan broj koji želimo da pomerimo (proizvoljno bita); opciono: drugi broj koji predstavlja broj mesta za koji pomeramo.
Izlazi: jedan broj koji je pomerena verzija ulaznog broja (isti broj bita kao i ulazni broj koji pomeramo).

Primer: 8-bitni pomerač koji pomera vrednost aritmetički za 2 mesta u desno.

Ulazni indeksi	7	6	5	4	3	2	1	0
Izlazni indeksi	7	7	7	6	5	4	3	2

```
oY <= iX(7) & iX(7) & iX(7 downto 2);
```

Još neki primeri u VHDL-u:

8-bitni pomerač koji pomera logički u levo za 2 mesta	`oY <= iX(5 downto 0) & "00";`
8-bitni pomerač koji pomera logički u desno za 2 mesta	`oY <= "00" & iX(7 downto 2);`
8-bitni pomerač koji pomera aritmetički u desno za 3 mesta	`oY <= iX(7) & iX(7) & iX(7) & iX(7 downto 3);`
8-bitni pomerač koji pomera logički u desno za 1 mesto i ubacuje bit sa ulaza iD	`oY <= iD & iX(7 downto 1);`

Sabirač

Funkcija: izlaz je aritmetički zbir dva ulaza.

Ulazi: dva broja (proizvoljno bita).
Izlazi: jedan broj koji ima isto bita kao i ulazni brojevi.

Primeri u VHDL-u:

Sabirač koji sabira dva 4-bitna broja i formira 4-bitni rezultat (bez prenosa)	`oY <= iA + iB;`
Sabirač koji sabira dva 4-bitna neoznačena broja i formira 5-bitni rezultat (sa prenosom)	`oY <= ('0' & iA) + ('0' & iB);`
Sabirač koji sabira 8-bitni broj iA sa 6-bitnim brojem iB i formira 8 bita rezultata (neoznačeni brojevi)	`oY <= iA + ("00" & iB);`
Sabirač koji sabira 8-bitni broj iA sa 6-bitnim brojem iB i formira 8 bita rezultata (označeni brojevi)	`oY <= iA + (iB(5) & iB(5) & iB);`