# A1. Auditorna vežba 1: Projektovanje sistema koji računa

## Motivacija

Na prvoj godini studija upoznali ste se sa fundamentalnim naukama, a i sa osnovama programiranja.

> S jedne strane: **fundamentalne nauke** (matematika, fizika, osnovi elektrotehnike).
> S druge strane: **osnovi programiranja** (programski jezici i strukture podataka, arhitektura računara).

Zadatak LPRS 1 predmeta je da napravi most znanja između gore navedenih grupa disciplina. Naučićemo kako se projektuje **računar** na kome možemo da primenimo znanja iz osnova programiranja, koristeći komponente koje ćemo osmisliti primenjujući znanja iz fundamentalnih nauka. Pošto je problem projektovanja računara težak, na ovom predmetu nam je cilj da naučimo kako se projektuje procesor.

★ Zadatak LPRS 1 predmeta je da naučimo kako se **projektuje procesor**, odn. sve potrebne gradivne komponente procesora.

? Podsetite se arhitekture računara. Kojih gradivnih komponenti procesora se sećate?

## Idejna skica procesora

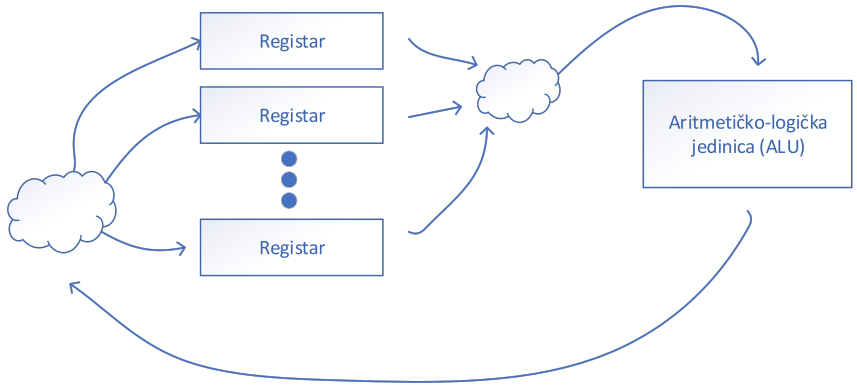
Podsetimo se nekih gradivnih komponenti procesora:

- **Registar** - služi da čuva vrednosti tokom izvršavanja programa (tj. kao prostor za promenljive);
- **Aritmetičko-logička jedinica** - služi da izvrši operaciju (sabiranje, množenje, pomeranje, itd.).

? Kako izgleda izvršavanje jedne aritmetičke instrukcije procesora, na primer, instrukcije tipa: `add eax, ebx` u asemblerskom jeziku ili `z = x + y` u C jeziku?

> *Aritmetička instrukcija podrazumeva da se vrednosti operanada (ulaznih promenljivih) nalaze u registrima. Te vrednosti treba da se proslede aritmetičko-logičkoj jedinici koja izvršava željenu operaciju. Nakon toga se rezultat iz aritmetičko-logičke jedinice upisuje u tačno jedan registar.*

Iz ovog primera dolazimo do sledeće idejne skice procesora, za potrebe izvršavanja jedne aritmetičke instrukcije.




Ova idejna skica nam zadaje sledeće zadatke koje ćemo rešavati tokom semestra:

1. Kako napraviti **sistem koji računa**, tj. izvršava aritmetičke i logičke operacije?
2. Kako rešiti detalje kao što su:
   a. Kako izvršiti **izbor** najčešće dva **operanda** koja želimo da pošaljemo ka ALU (desni oblačić)?
   b. Kako rezultat **upisati u tačno jedan registar** (levi oblačić)?
3. Kako napraviti **registar**?
4. Kako podržati izvršavanje **više od jedne instrukcije** i kontrolisati tok izvršavanja instrukcija?
5. Kako **integrisati** osmišljene komponente u jedan sistem - procesor?

Ovo su i teme pet auditornih vežbi koje su pred nama.

★ Današnji zadatak: kako napraviti sistem koji računa, tj. izvršava aritmetičke i logičke operacije?

## Predstava brojeva

### Predstava pozitivnih brojeva (i nule)

Da bismo mogli da napravimo sistem koji računa, prvi problem koji treba da rešimo je problem predstave brojeva u našem sistemu.

Neke mogućnosti su:

- mehanička predstava broja, sa kojom su radili prvi računari;
- električna predstava broja, koja je danas dominantna. Kod ove predstave, trenutna vrednost napona na žici (signalu) definiše brojnu vrednost koja je

trenutno predstavljena na toj žici (signalu).

Varijante električne predstave broja:

- analogna predstava - kontinualni opseg vrednosti na žici;
- digitalna predstava - diskretni opseg vrednosti na žici.

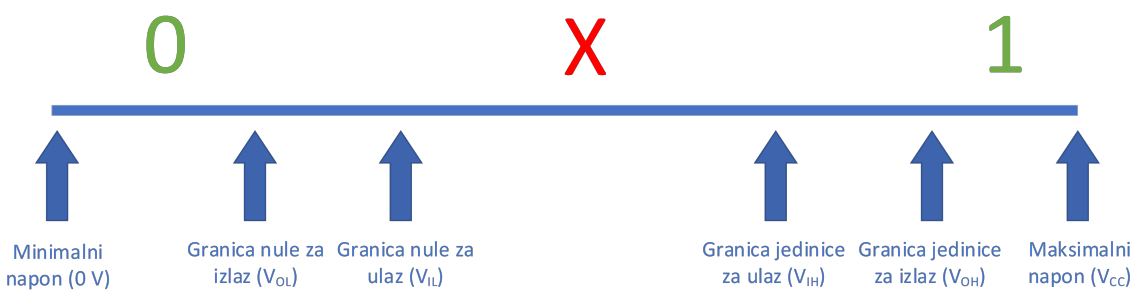
? Koliko diskretnih vrednosti da predstavimo na jednoj žici / signalu?

*Izbor koji je pobedilo je predstava **dve vrednosti** na jednoj žici. Ovo je najmanji broj vrednosti koji sa sobom nosi neku količinu informacije, pošto predstavom samo jedne vrednosti, data žica ne bi prenosila, odn. sadržala informaciju. Svaki veći broj vrednosti na jednoj žici donosi manju pouzdanost predstavljene vrednosti, a i zahteva složeniji sistem koji bi obrađivao takve signale. Te dve vrednosti zvaćemo nula (0) i jedan (1).*

? Koji interval napona da koristimo za predstavu vrednosti 0, a koji interval napona da koristimo za predstavu vrednosti 1?

*Kao što je uobičajeno u inženjerskom problemu, potrebno je naći kompromis.*
*Interval je poželjno da bude **što veći** da bi pouzdanost predstave bila što veća. Na primer, ukoliko naš signal na žici poremeti neki spoljni šum, želimo da on ne promeni vrednost tokom trajanja tog šuma (ovo se zove otpornost na šum).*
*Interval je poželjno da bude **što manji** da bi ukupna potrošnja našeg sistema bila manja (veći napon znači i veća potrošnja); što je interval manji, intervali za nulu i jedinicu mogu međusobno biti udaljeniji, pa samim tim i naš sistem otporniji na šum.*

Kompromisno rešenje je sledeće:




Definisaćemo **dve granice** za svaku vrednost na žici iz razloga što želimo da izlazna granica bude "strožija" od ulazne granice, kako bi naš sistem bio otporan na šum čak i ako se na izlazu formira vrednost koja je blizu gornje granice za nulu ili donje granice za jedinicu.

★ Sistem **na izlazu mora da formira vrednost u granicama za izlaz**, dok **na ulazu koristi ulazne granice** prilikom razumevanja vrednosti na žici (signalu).

? Kako onda predstaviti pozitivni broj veći od 1?

*Ukoliko jednu žicu iskoristimo za predstavu jedne cifre, kojih ukupno imamo dve (0 i 1), matematička osnova predstave pozitivnog broja pomoću više žica biće **binarni brojni sistem**.*

★ **Pozitivan broj (i nulu)** predstavljamo pomoću N žica, gde je N ukupan broj cifara potrebnih da se dati broj predstavi u binarnom brojnom sistemu. Vrednost broja čitamo na sledeći način (na primeru N = 4):

$$ABCD = A * 2^3 + B * 2^2 + C * 2^1 + D * 2^0$$

☐ **Primer 1**. Predstaviti broj 2020 u binarnom sistemu. Koliko žica nam treba za predstavu ovog broja?

*Rešenje.*
$2020 = 1024 + 512 + 256 + 128 + 64 + 32 + 4$. Dakle, binarna predstava broja je: $11111100100$.
Treba nam ukupno 11 žica za predstavu ovog broja.

Ubuduće ćemo jednu binarnu cifru ravnomerno zvati: žica, binarna cifra ili **bit** (pojam iz teorije informacija - količina informacije koju nosi poruka koja rešava neodređenost veličine 2, odn. nosi jednu od dve moguće vrednosti). Skup od jedne ili više žica, odn. bita zovemo **signal** koji, dakle, može biti jednobitni ili višebitni (dvobitni, trobitni, itd.).

## Predstava negativnih brojeva

? Kako da predstavimo broj $-1$?

Načini koje ste učili, ili ćete učiti, obuhvataju neku od sledećih predstava negativnih brojeva:

- predstava znak-moduo, kod koje dodajemo jedan bit za predstavu znaka (0 za plus, 1 za minus) i u nastavku prikažemo apsolutnu vrednost broja;
- predstava u prvom komplementu, koji predstavlja negaciju svih bita;
- predstava u drugom komplementu, koji je za jedan veći od prvog komplementa.

Međutim, ove predstave ovako definisane deluju ravnopravno i postavlja se pitanje koja predstava je najbolja. Da bismo se odlučili, moramo da definišemo kriterijum optimizacije ove odluke.

★ Donećemo odluku takvu da se **aritmetičke operacije** sa negativnim brojeva **izvršavaju na isti način**, odn. pomoću istog postupka kao i sa pozitivnim brojevima. Tada ćemo imati lakši posao prilikom pravljenja sistema koji izvršavaju ove operacije, pošto će isti sistem moći da izvršava operacije i nad pozitivnim i nad negativnim brojevima.

Pre nego što odgonetnemo koji način predstave negativnog broja je optimalan po ovom kriterijumu, podsetićemo se kako sabiramo binarne brojeve.

## Sabiranje binarnih brojeva

? Uradite ručno sabiranje sledećih binarnih brojeva:

```
  1011
 +0110
--------
  10001
```

Posmatrajmo sada postupak sabiranja binarnih brojeva **korak po korak**.

- U prvom koraku, sabrali smo krajnje desne cifre (1 + 0), **zapisali rezultat sabiranja** (1) i **primetili da ne treba ništa da prenesemo** (0);
- U koraku 2, **sabrali smo prenos i naredne dve cifre** (0 + 1 + 1), zapisali desnu cifru rezultata sabiranja (0) i zapamtili prenos (1);
- U koraku 3, sabrali smo prenos i naredne dve cifre (1 + 0 + 1), zapisali desnu cifru rezultata sabiranja (0) i zapamtili prenos (1);
- U koraku 4, sabrali smo prenos i naredne dve cifre (1 + 1 + 0), zapisali desnu cifru rezultata sabiranja (0) i zapamtili prenos (1);
- Na kraju, zapisali smo poslednji prenos kao petu cifru rezultata (1).

Zaključujemo da je sabiranje binarnih brojeva veoma repetitivna operacija, što će nam pomoći prilikom formiranja sistema koji sabira. Za sada je dovoljno što smo se podsetili samog postupka sabiranja, pa se vraćamo na predstavu negativnih brojeva.

## Predstava negativnih brojeva (nastavak)

? Kako da predstavimo broj $-1$ ukoliko na raspolaganju imamo 4 bita za predstavu jedne brojne vrednosti?

Jedna osobina broja $-1$ je ta da, kada na njega dodamo $+1$, kao rezultat dobijamo $0$. Dakle, šta treba da bude predstava broja $-1$ u sledećem primeru sabiranja?

```
  ABCD  (-1)
 +0001  (+1)
--------
  0000   (0)
```

Ukoliko smo ograničeni na 4 bita, peti bit rezultata (prenos) neće biti vidljiv. Primećujemo da je jedina kombinacija bita koja može da se ubaci u prethodnu jednačinu: ABCD = 1111. Dakle, jedina prirodna predstava broja $-1$, za koju će "raditi" operacija sabiranja onako kako smo navikli, je **1111**.

Na sličan način možemo zaključiti da je jedina prirodna predstava broja -2, za koju će "raditi" operacija sabiranja onako kako smo navikli, kombinacija **1110** (jer je 1110 + 0001 = 1111, a takođe: 1110 + 0010 = (1)0000 [prenos gubimo!]).

Dolazimo do sledećeg zaključka o načinu predstave celih brojeva kod kojeg će operacija sabiranja biti identična kao i za neoznačene brojeve:

```
... 1100 1101 1110 1111 0000 0001 0010 0011 0100 ...
     -4   -3   -2   -1    0   +1   +2   +3   +4
```

Ova predstava celog broja se zove **predstava broja u drugom (II) komplementu**. Postavlja se pitanje gde napraviti granicu sa leve i desne strane, pošto ukoliko sa leve strane dođemo do broja $1000$ on bi predstavljao $-8$, a ukoliko sa desne strane dođemo do broja $1000$ on bi predstavljao $+8$.

Pošto je veoma korisno da nam neki bit predstavlja znak, da bismo lako znali da li je broj pozitivan ili negativan, iz gornje brojne prave možemo zaključiti da bi taj bit mogao biti krajnji levi bit. Dakle, svaki broj koji počinje sa 1 ćemo smatrati negativnim, odn. brojem "s leve strane" brojne prave, a svaki broj koji počinje sa 0 ćemo smatrati pozitivnim, odn. brojem "s desne strane" brojne prave. Kompletirajmo brojnu pravu za predstavu ograničenu na 4 bita:

```
1000 1001 1010 1011 1100 1101 1110 1111 0000 0001 0010 0011 0100 0101 0110 0111
 -8   -7   -6   -5   -4   -3   -2   -1    0   +1   +2   +3   +4   +5   +6   +7
```

Princip se lako proširuje na predstavu u više bita: nula je uvek u sredini, jer želimo da predstava obuhvati približno jednak broj pozitivnih i negativnih brojeva, a ostale brojeve definišemo na isti način, dodajući 1 s desne strane, odn. oduzimajući 1 s leve strane.

**Primer 2.** Koji opseg brojeva možemo predstaviti u predstavi u drugom komplementu, ako na raspolaganju imamo 5 bita?

*Rešenje.*

Ako pogledamo gornju brojnu pravu, možemo zaključiti da će najmanji negativan broj biti 10000, koji je tačno 16 mesta u levo od 00000, dakle predstavlja broj -16. Najveći pozitivan broj će biti 01111 koji predstavlja +15. Dakle, opseg 5-bitnih označenih brojeva je [-16, +15].

Može se pokazati da između broja X i njemu suprotnog broja -X u predstavi drugog komplementa važi sledeća jednakost:

$$-\boldsymbol{X} = \boldsymbol{not}(\boldsymbol{X}) + \mathbf{1}$$

Dekadna vrednost pozitivnog broja se lako otkriva, na isti način kao i za neoznačeni broj, međutim vrednost negativnog broja ne možemo lako otkriti. Gore navedena formula nam ovo olakšava, pošto pomoću nje lako možemo da otkrijemo suprotan broj negativnom broju, koji je pozitivan i lako možemo da ga pročitamo.

**Primer 3.** Koristeći formulu $-X = not(X) + 1$, po potrebi, otkrijte vrednost sledećeg 8-bitnog označenog binarnog broja: $11100101$.

*Rešenje.*

Negativni broj ne možemo lako da pročitamo, pa ćemo prvo da otkrijemo njemu suprotan broj ($-X$) primenjući formulu; prvo negiramo sve bite, pa

dodamo 1: $11100101 \rightarrow 00011010 \rightarrow 00011011$. Ovaj broj možemo lako pročitati kao binarni broj vrednosti $+27$. Dakle, pošto je $-X = 27$, naš traženi broj je $X = -27$.

**Primer 4.** Koristeći formulu $-X = not(X) + 1$, po potrebi, otkrijte vrednost sledećeg 8-bitnog označenog binarnog broja: $00001010$.

*Rešenje.*

Ovo je pozitivan broj, tako da ga možemo odmah pročitati! Nema potrebe za računanjem njemu suprotnog. Vrednost broja je $X = +10$.

Ubuduće ćemo za predstavu označenih brojeva uvek podrazumevati predstavu u drugom komplementu, ukoliko to drugačije ne naglasimo.

### Predstava ostalih brojeva i podataka

Sledi kratak pregled predstave brojeva i nekih ostalih podataka u računarskom sistemu (u zagradi je primer tipa promenljive u C jeziku):

- neoznačeni celi broj (**unsigned int**): obična binarna predstava bez znaka; svaka cifra je neki stepen dvojke;
- označeni celi broj (**int**): predstava u drugom komplementu;

- racionalni broj:
  - može biti predstavljen u predstavi sa nepokretnim zarezom, kod koje se cifre desno od zareza posmatraju na isti način kao i u dekadnom sistemu, samo sa osnovom 2; na primer:

    $1011.101 = 2^3 + 2^1 + 2^0 + 2^{-1} + 2^{-3} = 11.625$

    (ova predstava takođe postoji u neoznačenoj i označenoj varijanti, koju možemo konstruisati na isti način kao i kod celih brojeva - u drugom komplementu).

  - može biti predstavljen i u predstavi sa pokretnim zarezom, tzv. *floating-point*. Ovo je način predstave u C jeziku (**float** i **double**). https://en.wikipedia.org/wiki/Single-precision_floating-point_format

- realni broj: najbližom aproksimacijom koja staje u onoliko bita koliko imamo na raspolaganju (ovo važi i za mnoge racionalne brojeve koji imaju neograničeno decimala ili više decimala nego što je na raspolaganju);

- tekst, slika, zvuk, itd.: pomoću brojeva! Svi podaci u računarskom sistemu se predstavljaju brojem, u binarnoj predstavi.

## Transformacija podataka (brojeva) u računarskom sistemu

Kao primer transformacije, odn. obrade podataka (brojeva) u računarskom sistemu uzećemo operaciju sabiranja. Ponovićemo primer i zaključak od ranije.

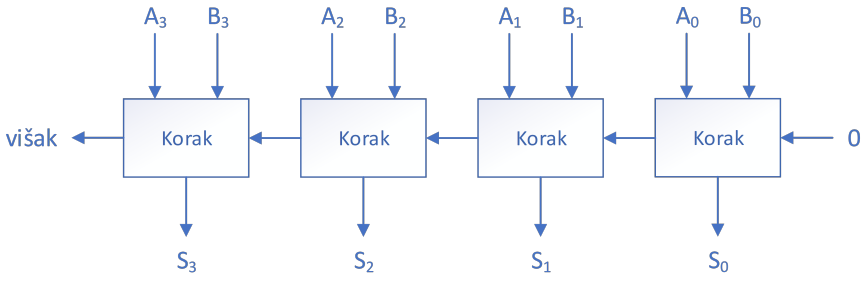
```
  1011
 +0110
--------
 10001
```

Posmatrajmo sada postupak sabiranja binarnih brojeva **korak po korak**.

- U prvom koraku, sabrali smo krajnje desne cifre (1 + 0), **zapisali rezultat sabiranja** (1) i **primetili da ne treba ništa da prenesemo** (0);
- U koraku 2, **sabrali smo prenos i naredne dve cifre** (0 + 1 + 1), zapisali desnu cifru rezultata sabiranja (0) i zapamtili prenos (1);
- U koraku 3, sabrali smo prenos i naredne dve cifre (1 + 0 + 1), zapisali desnu cifru rezultata sabiranja (0) i zapamtili prenos (1);
- U koraku 4, sabrali smo prenos i naredne dve cifre (1 + 1 + 0), zapisali desnu cifru rezultata sabiranja (0) i zapamtili prenos (1);
- Na kraju, zapisali smo poslednji prenos kao petu cifru rezultata (1).

Primećujemo da se sabiranje dešava u 4 gotovo identična koraka. Jedini koraci koji se razlikuju su prvi i poslednji. Prvi korak lako možemo da izjednačimo sa ostalim koracima tako što **podrazumevamo da je prenos prvom koraku uvek 0**, odn. da sabiramo nulu i krajnje desne cifre (u gornjem primeru: 0 + 1 + 0). Poslednji korak možemo zanemariti, jer ako smo ograničeni na 4 bita ni nemamo prostor za peti bit rezultata (u ovom primeru, smatrali bismo da rezultat izlazi iz opsega koji možemo predstaviti unutar 4 bita).

Dakle, sistem koji sabira bismo mogli da predstavimo sledećom blok šemom:




Svaki korak je identičan i predstavlja **računanje dve vrednosti** kao funkciju **tri ulazne vrednosti**. Sistemski gledano, svaki korak ima:

- **ulaze**: tri vrednosti (dve cifre koje se sabiraju i prenos iz prethodnog stepena), svaka vrednost može biti 0 ili 1;
- **izlaze**: dve vrednosti (koje zajedno predstavljaju 2-bitnu vrednost zbira svih ulaznih vrednosti, desnu cifru pišemo dole kao rezultat, a levu cifru prenosimo u naredni stepen), svaka vrednost na izlazu može biti 0 ili 1.

Dakle, sistem koji sabira ćemo napraviti **kombinacijom koraka (sistema) koji računaju matematičke funkcije sledećeg oblika:**

$f\colon \quad \{0,1\}^n \rightarrow \{0,1\}$ odn. funkcije $n$ promenljivih nad skupom $\{0,1\}$, koje kao rezultat imaju broj u skupu $\{0,1\}$.

Ovo su nam dobro poznate funkcije iz **Bulove algebre**.

★ Time smo uspeli da problem računanja vrednosti aritmetičke funkcije, kao što je sabiranje, raščlanimo na elementarne korake koji prestavljaju problem računanja vrednosti logičke (Bulove) funkcije.

Dakle, u nastavku ćemo dati fokus na **projektovanju sistema koji računa logičke funkcije**, čijom kombinacijom možemo onda dobiti sistem koji računa aritmetičke funkcije.

## Projektovanje sistema koji računaju logičke funkcije

Sistem koji računa logičke (Bulove) funkcije će, u opštem slučaju, imati N ulaza i M izlaza. Svaki izlaz će biti posebna logička funkcija ovih N ulaza, tako da svaki izlaz možemo posmatrati nezavisno i fokusirati se na realizaciju sistema koji ima jedan izlaz, koji predstavlja funkciju N ulaza:

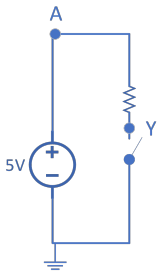
$f\colon \quad \{0,1\}^n \rightarrow \{0,1\}$

Bulova algebra nas uči da **svaku logičku funkciju** od proizvoljnog broja promenljivih možemo predstaviti kao **kombinaciju funkcija iz minimalnog skupa funkcija** neophodnih za definisanje cele Bulove algebre. Jedan od minimalnih skupova Bulove algebre sadrži sledeće tri funkcije: logičku konjunkciju ("i", AND), logičku disjunkciju ("ili", OR) i logičku negaciju ("ne", NOT).

★ Kada napravimo sisteme koji računaju tri osnovne logičke funkcije (AND, OR, NOT), njihovom kombinacijom možemo napraviti sistem koji računa bilo koju logičku funkciju, sa proizvoljno velikim brojem ulaznih promenljivih.

### Projektovanje sistema koji računa logičku negaciju (NOT)

Sistem koji računa logičku negaciju treba da, kada na ulazu dobije niski napon (0), na izlazu formira visok napon (1) i obrnuto. Posmatrajmo sledeće električno kolo:




Analizirajmo ponašanje ovog kola kao funkciju stanja na prekidaču. Potencijal tačke A je +5V, pošto se ona nalazi na pozitivnom kraju baterije koja pravi razliku potencijala +5V u odnosu na negativni kraj koji smatramo potencijalnom nulom (što je označeno simbolom zemlje na električnoj šemi).

? Koliki je potencijal tačke Y ako je prekidač isključen?

*Ako je prekidač isključen, u kolu nema struje, pa je potencijal tačke Y jednak potencijalu tačke A, odn. +5V.*

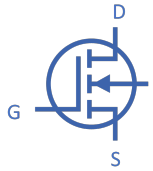
? Koliki je potencijal tačke Y ako je prekidač uključen?

*Ako je prekidač uključen, tačka Y je kratko spojena na zemlju, pa je njen potencijal 0V.*

Ukoliko stanje prekidača posmatramo kao ulaznu logičku vrednost (0 - isključen, 1 - uključen), a potencijal tačke Y kao izlaznu logičku vrednost (0, 1), sledeća istinitosna tablica opisuje ponašanje električnog kola:

| Ulaz | Izlaz |
|---|---|
| 0 | 1 |
| 1 | 0 |

Dakle, ovo električno kolo **računa funkciju logičke negacije**. Prekidač u računarskom sistemu ne može biti mehanički, pa se umesto prekidača koristi poluprovodnička komponenta **tranzistor**. Jedan od najčešće korišćenih tranzistora u današnjim logičkim kolima je **MOSFET**.

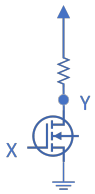



G - *gate*
S - *source*
D - *drain*

Iznad je dat jedan od simbola koji se koristi za ovu komponentu (može i bez srednje strelice). Ponašanje ove komponente je sledeće:

- Ukoliko je napon između G i S manji od nekog pozitivnog praga (tipično je prag manji od 1V), ova komponenta **ne provodi** između pinova D i S.
- Ukoliko je napon između G i S veći od tog praga, komponenta **provodi** između pinova D i S.

Implementacija logičkog kola sa MOSFET tranzistorom umesto prekidača je sledeća:



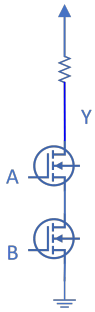

| Ulaz | Izlaz |
|---|---|
| 0 | 1 |
| 1 | 0 |

Ponašanje kola je isto kao i sa prekidačem:
- ako je na ulazu X niska vrednost (0), tranzistor ne provodi, pa je Y spojen na Vcc (1);
- ako je na ulazu X visoka vrednost (1), tranzistor provodi, pa je Y spojen na (0).

U stvarnoj realizaciji, umesto otpornika se koristi malo drugačiji MOSFET tranzistor, odn. onaj koji se ponaša suprotno od tranzistora u donjem delu kola. Razlog za to je taj da, kada "donji" tranzistor provodi, "gornji" tranzistor ne provodi, i obrnuto. Ponašanje takvog kola je identično ovom. Tehnologija koja koristi oba tipa MOSFET-a (N-MOSFET koji se koristi u "donjem" delu kola i P-MOSFET koji se koristi u "gornjem" delu kola) se zove CMOS tehnologija i danas je dominantna tehnologija izrade logičkih kola. (Sliku CMOS invertora imate u beleškama sa predavanja.)

## Projektovanje sistema koji računa logičku konjunkciju (AND)

Pokušajmo da MOSFET tranzistore povežemo redno, u nadi da ćemo dobiti ponašanje koje predstavlja logičku konjunkciju, jer redna veza prekidača odgovara logičkoj konjunkciji, pošto struja može da teče kroz rednu vezu prekidača samo u slučaju kada su oba prekidača uključena.



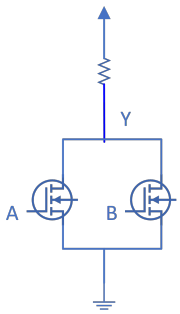

| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Ponašanje ovog kola se lako može analizirati:
- ako je **bar jedan ulaz na niskoj vrednosti**, bar jedan tranzistor neće provoditi, pa će samim tim Y biti povezan na Vcc (1);
- ako su **oba ulaza na visokoj vrednosti**, oba tranzistora provode i Y je spojen na (0).

Međutim, ovo nije logička konjunkcija, nego **negacija konjunkcije (NAND, NI)**, jedno od osnovnih kola CMOS tehnologije. Da bismo dobili logičku konjunkciju, potrebno je izlaz ovog kola povezati na izlaz invertora (NOT). Negacija negacije konjunkcije daće nam potrebnu logičku konjunkciju (AND).

## Projektovanje sistema koji računa logičku disjunkciju (OR)

Pokušajmo sada da MOSFET tranzistore povežemo paralelno, u nadi da ćemo dobiti ponašanje koje predstavlja logičku disjunkciju, jer paralelna veza prekidača odgovara logičkoj disjunkciji, pošto struja može da teče kroz paralelnu vezu prekidača u slučaju kada je bar jedan prekidač uključen.




| A | B | Y |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

Ponašanje ovog kola se lako može analizirati:
- ako su **oba ulaza na niskoj vrednosti**, bar tranzistori neće provoditi, pa će samim tim Y biti povezan na Vcc (1);
- ako je **bar jedan ulaz na visokoj vrednosti**, bar jedan tranzistor provodi i Y je spojen na (0).

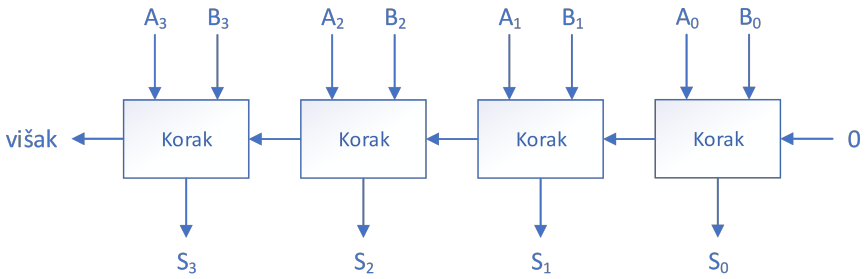
Međutim, ovo nije logička disjunkcija, nego **negacija disjunkcije (NOR, NILI)**, jedno od osnovnih kola CMOS tehnologije. Da bismo dobili logičku disjunkciju, potrebno je izlaz ovog kola povezati na izlaz invertora (NOT). Negacija negacije disjunkcije daće nam potrebnu logičku disjunkciju (OR).

- U realnoj realizaciji u CMOS tehnologiji, u "gornjem" delu logičkog kola se ne nalaze otpornici, nego P-MOSFET-i koji su povezani u komplementarnoj logici, tako da "gornji" deo kola provodi onda kada "donji" deo ne provodi i obrnuto. To znači da, ako su tranzistori u "donjem" delu kola povezani redno, u "gornjem" delu kola treba da budu povezani paralelno i obrnuto.

- Pošto smo implementirali sisteme koji realizuju minimalni skup logičkih operacija, sada imamo način da projektujemo sistem koji računa proizvoljnu logičku funkciju (to nam garantuje matematika Bulove algebre). Samim tim, imamo sve potrebne gradivne komponente da završimo naš sabirač.

# Projektovanje sistema koji sabira

Podsetimo se šeme sistema koji sabira:




Elementarni korak je **sabiranje tri logičke promenljive**, odn. tri bita i formiranje rezultata koji sadrži dva bita; jedan se piše kao deo rezultata (S) i jedan se prenosi u naredni stepen (P). Formirajmo istinitosnu tablicu koja definiše kako elementarni korak treba da se ponaša za svaku kombinaciju ulaza, odn. koje logičke funkcije predstavljaju izlazi S i P svakog elementarnog koraka.

| A | B | C | P | S |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

Izlaz možemo da čitamo kao 2-bitni broj P-S i on treba da predstavlja zbir tri 1-bitne vrednosti na ulazima A, B i C.

Bulova algebra nas uči kako da formiramo jednačine za svaki izlaz, pomoću SDNF.

$P = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$S = \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + ABC$

? Kako se formira SDNF oblik jednačine logičke funkcije iz istinitosne tablice?

*SDNF se formira kao logički zbir (OR) logičkih proizvoda (AND). Za svaku kombinaciju ulaza za koju izlaz funkcije ima vrednost 1, formira se proizvod od svih promenljivih na ulazu, tako da promenljiva u proizvod ulazi negirana ako joj je u toj kombinaciji ulaza vrednost 0, odn. direktno ako joj je vrednost 1. Na primer, za kombinaciju ulaza 011 proizvod je:* $\bar{A} * B * C$. Svi proizvodi za slučajeve u kojima funkcija ima vrednost 1 ulaze u logičku disjunkciju (OR).

Koristeći tehnike minimizacije logičkih funkcija sa predavanja, minimizirajte svaku od gore navedenih logičkih funkcija.

! Koristeći simbole logičkih kola koje ste naučili na računarskim vežbama i gore navedene jednačine, pokušajte da napravite implementaciju elementarnog koraka sabiranja i proverite njegovo ponašanje u simulaciji.

Kao dodatni izazov, pokušajte da povežete 4 elementarna koraka prema gore nacrtanoj šemi u kompletan 4-bitni sabirač!