УНИВЕРЗИТЕТ "УНИОН-НИКОЛА ТЕСЛА"
ПОСЛОВНИ И ПРАВНИ ФАКУЛТЕТ БЕОГРАД

Osnovi programskih jezika

- Programiranje, jezici i prevodioci-

Docent prof. dr Borivoje M. Milošević

Osnove softvera

Program (softver):

- set instrukcija, nalazi se u memoriji računara i izvršava potrebne instrukcije za njegov rad

Osnovne kategorije programa (softvera):

Sistemski softver:

- Upravlja resursima računara: hardver, memorija, adrese, ...

Aplikativni softver:

- Alat za pomoć računarskim korisnicima u rešavanju problema iz realnog sveta

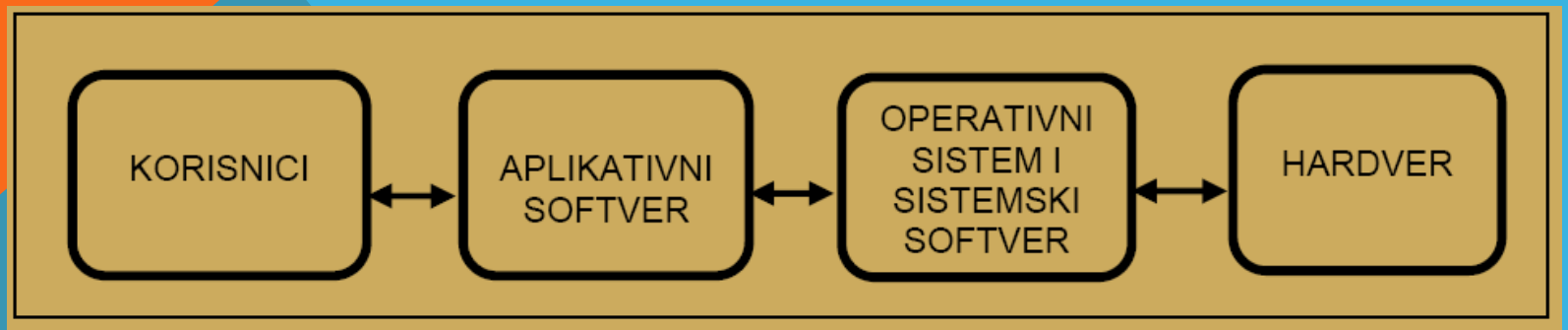
Kompajleri i drugi programi za prevođenje:

- Omogućavaju programerima kreiranje novog softvera

POJAM I KLASIFIKACIJA SOFTVERA

Bez softvera hardver informacionih sistema je praktično neupotrebljiv. Osnovna karakteristika računarskog sistema je integralna povezanost mašinske i programske podrške, odnosno hardvera i softvera. Iz tog razloga je razvoj hardvera pratio i razvoj softvera.

Softer obuhvata sve nematerijalne, logičke komponente koje su neophodne za pravilan i kvalitetan rad i korišćenje računara. Softver čine programi koji s jedne strane kontrolišu funkcionisanje računara i ostalih delova informacionog sistema, a sa druge strane vrše obradu podataka. Softver dakle možemo podeliti na sistemski i aplikativni softver.



Aplikativni softver

Aplikativni softver pomaže pri obavljanju poslova koje korisnik želi obaviti uz pomoć računara. Aplikativni softver čine programi i programski paketi koji su izgrađeni da razreše konkretne probleme u mnogim oblastima. To su programi:

- za obradu teksta,
- rad sa bazama podataka,
- programi za prezentaciju, ili pak
- programi za evidenciju i obračun plata,
- evidenciju dobavljača,
- evidenciju kupaca,
- finansijsko knjigovodstvo,
- upravljanje proizvodnjom,
- upravljanje zalihama ili
- statistička istraživanja.

Znači, za svaku namenu računara mora postojati poseban aplikativni softver. Aplikativni softver je pre svega orijentisan ka korisniku, pokušavajući da na najlakši način zadovolji njegove potrebe. Poslom izrade aplikacija se bave programeri jer se zahtevi korisnika razlikuju od slučaja do slučaja.

Sistemski softver

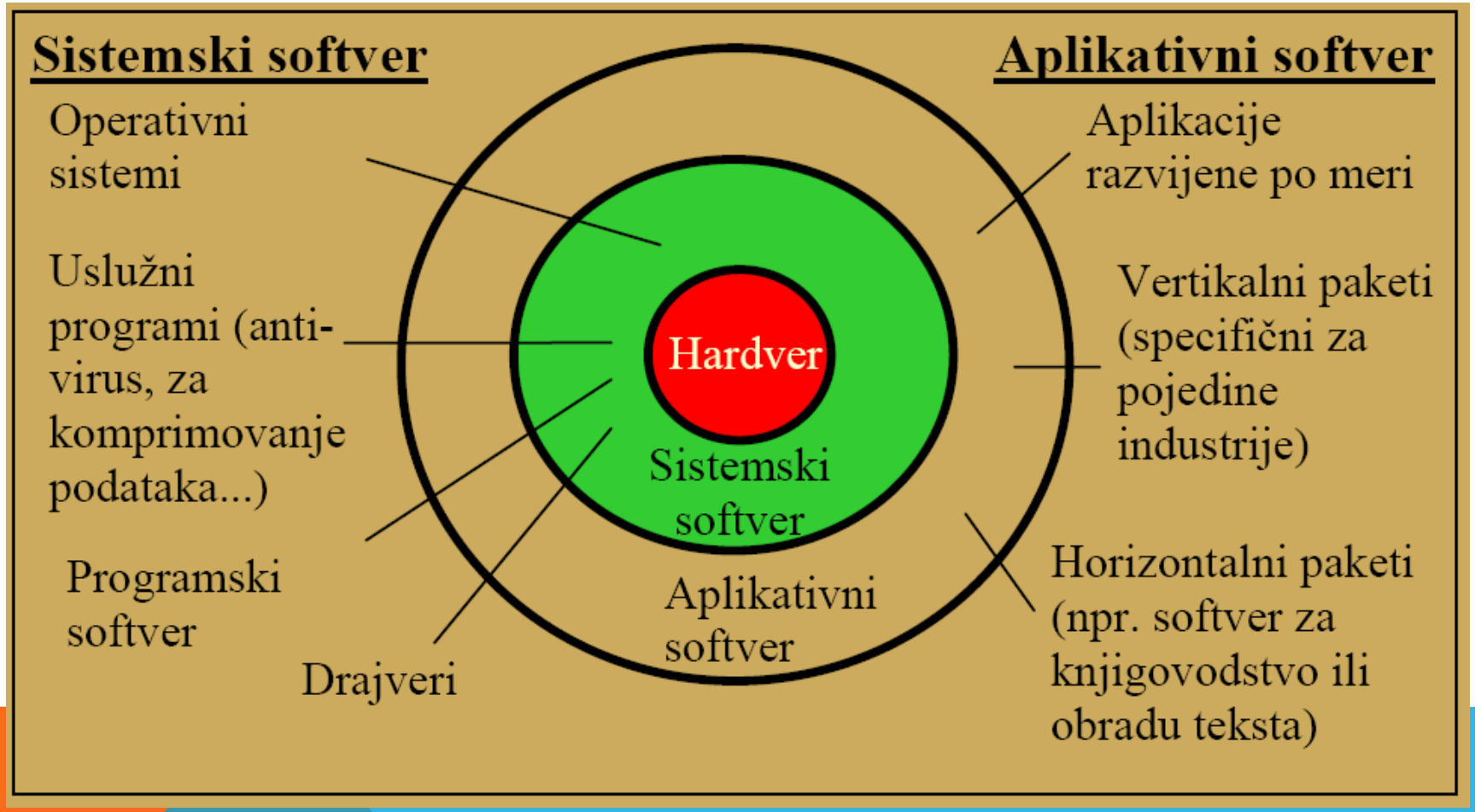
Sistemski softver čine programi koji upravljaju računarskim sistemom i pomažu programiranje aplikacija. Pod sistemskim softverom podrazumevaju se:

• **Operativni sistemi** – neophodni za pokretanje i rad računara.

• **Uslužni programi** – omogućavaju korisniku računara upravljanje i kontrolu lokacija podataka i hardverskih komponenata

• **Programski softver** – alat za rešavanje zadataka na odgovarajući način, pomoću kojeg se uz implementaciju i prilagođavanje, smeštanjem u određeni modul izrađuje proizvod – aplikativni softver. Za pisanje programa neophodno je koristiti neki od programskih jezika. Neki od najpopularnijih programskih jezika danas su Visual Basic, C++, C#, Perl, Java i drugi.

• **Drajveri** – softver pomoću kojih kompjuter kontroliše periferne jedinice. Drajveri softverski povezuju hardverske komponente sa računarom. Jednom instaliran drajver, računar koristi "iza scene" za komunikaciju sa pojedinom hardverskom jedinicom.


Sistemski softver
Operativni sistemi
Uslužni programi (anti-virus, za komprimovanje podataka...)
Programski softver
Drajveri
Aplikativni softver
Aplikacije razvijene po meri
Vertikalni paketi (specifični za pojedine industrije)
Horizontalni paketi (npr. softver za knjigovodstvo ili obradu teksta)
Hardver
Sistemski softver
Aplikativni softver

Životni ciklus razvoja softvera

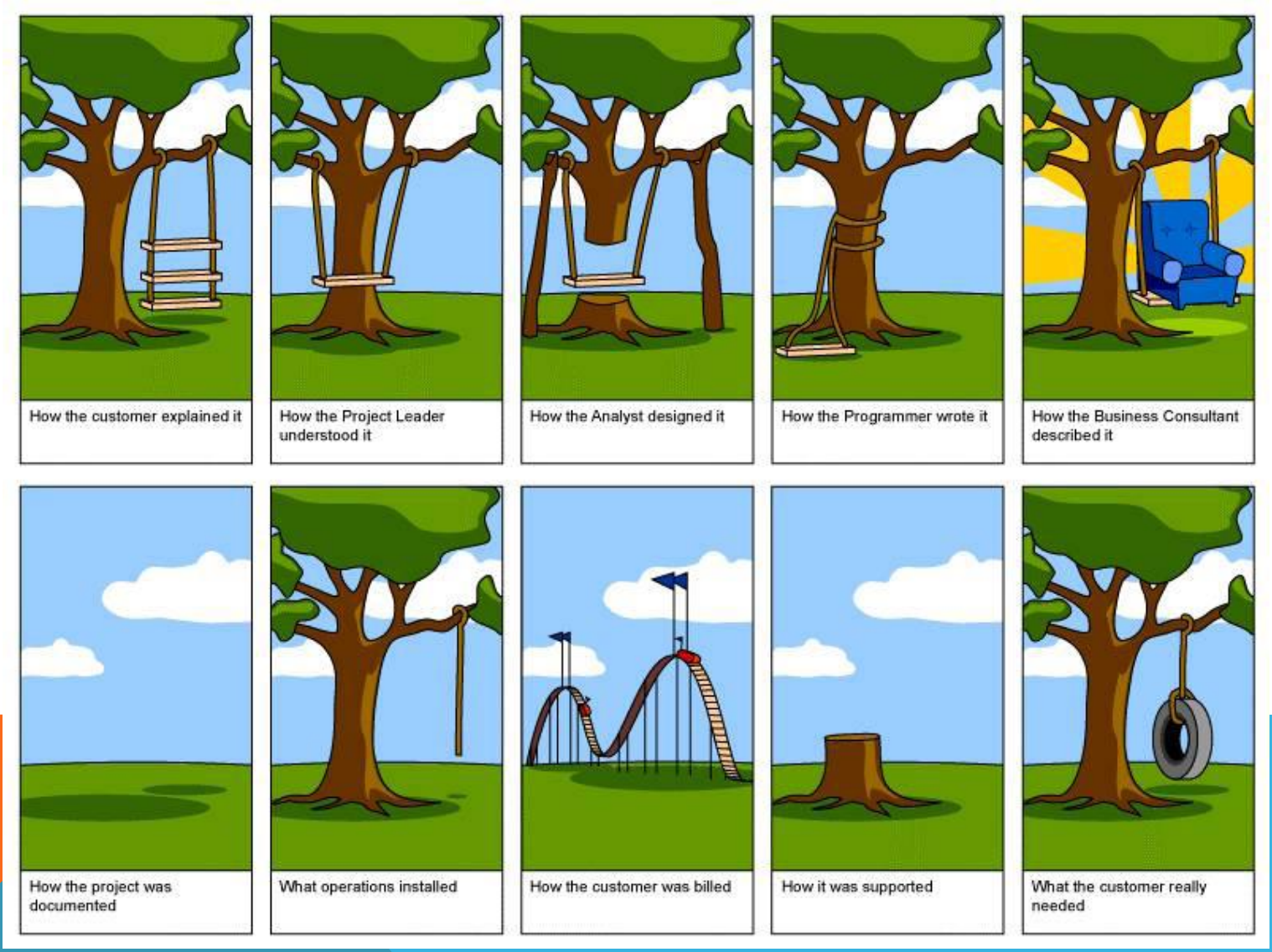
Pod razvojem softvera često se ne misli samo na neposredno pisanje programa, već i na procese koji mu prethode i slede. U tom, širem smislu, razvoj softvera naziva se i životni ciklus razvoja softvera. Razvoj softvera razlikuje se od slučaja do slučaja, ali u nekoj formi obično ima sledeće faze i podfaze:

- Planiranje: Ova faza obuhvata prikupljanje i analizu zahteva od naručioca softvera, razrešavanje nepotpunih, višesmislenih ili kontradiktornih zahteva i kreiranje precizne specifikacije problema i dizajna softverskog rešenja.

- Realizacija: Ova faza obuhvata implementiranje dizajniranog softverskog rešenja u nekom konkretnom programskom jeziku. Analizom efikasnosti i ispravnosti proverava se pouzdanost i upotrebljivost softverskog proizvoda, a za naručioca se priprema i dokumentacija.

- Eksploatacija: Ova faza počinje nakon što je ispravnost softvera adekvatno proverena i nakon što je softver odobren za upotrebu.

Faze razvoja softvera ilustrovane na šaljiv način



Planiranje

Poslovna analiza u fazi planiranja bavi se, pre svega, preciznom postavkom i specifikacijom zahteva, dok se modelovanje i dizajn bave razradom projekta koji je definisan u fazi analize.

U fazi planiranja često se koriste različite dijagramske i algoritamske tehnike i specijalizovani alati koji podržavaju kreiranje ovakvih dijagrama (tzv. CASE alati – Computer Aided Software Engineering).

Procesom planiranja strateški rukovodi arhitekta čitavog sistema (engl. enterprise architect, EA). Njegov zadatak je da napravi opšti, apstraktan plan svih procesa koji treba da budu softverski podržani.

Analiza i specifikacija problema

Proces analize i specifikacije problema obično sprovodi poslovni analitičar (engl. business analyst, BA), koji nije nužno informatičar, ali mora da poznaje relevantne poslovne ili druge procese u okviru zahtevanog rešenja.

Kada se softver pravi po narudžbini, za poznatog kupca, u procesu analize i specifikacije problema vrši se intenzivna komunikacija poslovnog analitičara sa naručiocioma, krajnjim korisnicima ili njihovi predstavnicima. Kada se softver pravi za nepoznatog kupca, često u kompanijama ulogu naručioca preuzimaju radnici zaposleni u odeljenju prodaje ili marketinga (koji imaju ideju kakav proizvod bi kasnije mogli da prodaju).

U komunikaciji poslovnog analitičara sa naručiocima, često se najpre vrši analiza postojećih rešenja (na primer, postojećeg poslovnog procesa u kompaniji koja uvodi informacioni sistem) i razmatraju se mogućnosti njihovog unapredenja uvodenjem novog softvera.

Naručioci često nemaju informatičko obrazovanje, pa njihovi zahtevi koje softver treba da zadovolji mogu da budu neprecizni ili čak i kontradiktorni.

Modelovanje rešenja

Modelovanje rešenja obično sprovodi arhitekta rešenja (engl. solution architect, SA), koji mora da razume specifikaciju zahteva i da je u stanju da izradi matematičke modele problema i da izabere adekvatna softverska rešenja, npr. programski jezik, bazu podataka, relevantne biblioteke, strukture podataka, algoritamska rešenja, itd.

Zbog čega?

Dizajn softverskog rešenja

U procesu dizajniranja, arhitekta softvera (engl. software architect) vrši preciziranje rešenja i opisuje arhitekturu softvera (engl. software architecture) – celokupnu strukturu softvera i načine na koje ta struktura obezbeduje integritet sistema i željeni ishod projekta (ispravan softver, dobre performanse, poštovanje rokova i uklapanje u planirane troškove).

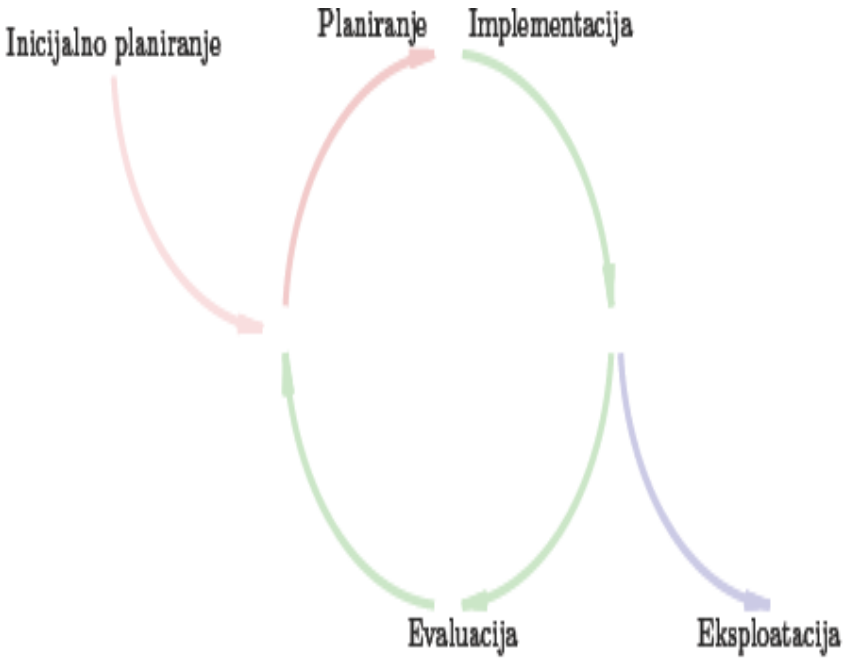
Neke od osnovnih tema koje se razmatraju u okviru dizajna softvera su:

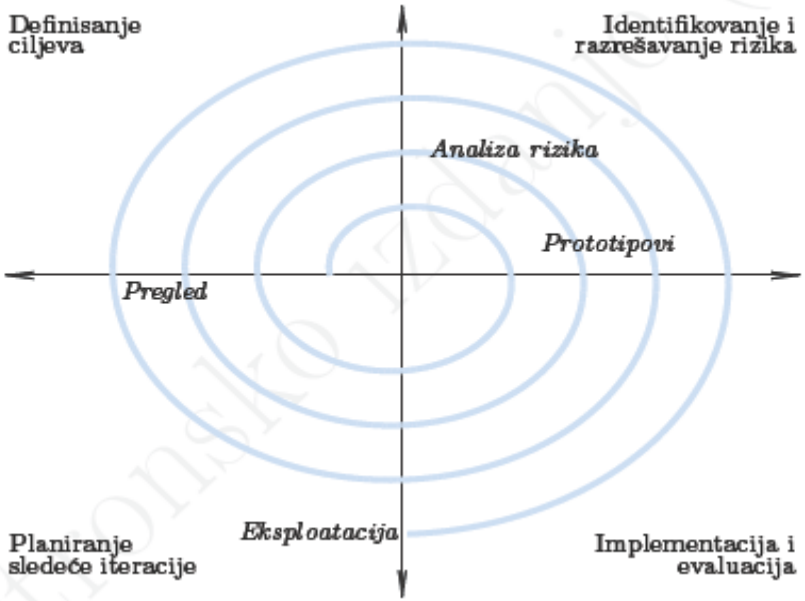
- Apstrahovanje (engl. abstraction) – apstrahovanje je proces generalizacije kojim se odbacuju nebitne informacije tokom modelovanja nekog entiteta ili procesa.
- Profinjavanje (engl. refinement) – profinjavanje je proces razvoja programa odozgo-naniže.
- Dekompozicija (engl. decomposition) – cilj dekompozicije je razlaganje na komponente koje je lakše razumeti, realizovati i održavati.
- Modularnost (engl. modularity) – softver se deli na komponente koje se nazivaju moduli.

Metodologije razvoja softvera

Metodologija vodopada: U strogoj varijanti ove tradicionalne metodologije, opisane prvi put još pedesetih godina prošlog veka, na sledeću fazu u razvoju softvera prelazi se tek kada je jedna potpuno završena.

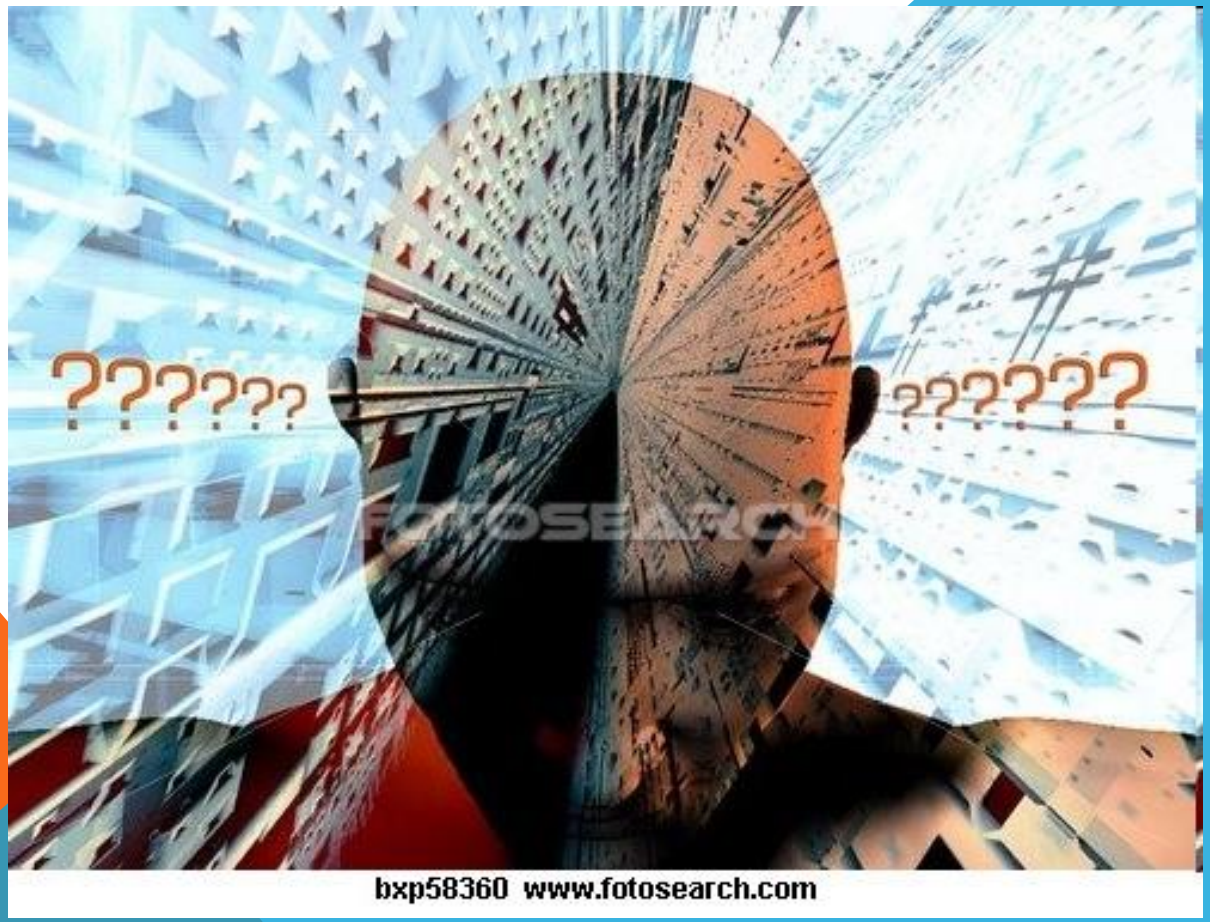
Iterativna i spiralna metodologija:





TUNEL ??????

PROGRAMIRANJE



PRINCIPI PISANJA PROGRAMA

Programi napisani na višem programskom jeziku sredstvo su komunikacije izmedu čoveka i računara, ali i izmedu ljudi samih. Razumljivost, čitljivost programa, iako nebitna za računar, od ogromne je važnosti za kvalitet i upotrebljivost programa.

Naime, u održavanje programa obično se uloži daleko više vremena i truda nego u njegovo pisanje, a održavanje sistema često ne rade oni programeri koji su program napisali.

Pored toga, razumljivost programa omogućava lakšu analizu njegove ispravnosti i složenosti. Preporuke za pisanje često nisu kruta pravila, već predstavljaju samo ideje kojima se treba rukovoditi u pisanju programa, u aspektima formatiranja, modula, imenovanja promenljivih i funkcija itd.

VIZUALNI ELEMENTI PROGRAMA

Prva ideja o programu formira se na osnovu njegovog izgleda – njegovih vizualnih elemenata, kao što su broj linija u datoteci, broj karaktera u liniji, tabulisanje, grupisanje linija i slično. Vizualni elementi programa i njegovo formatiranje često su od ključne važnosti za njegovu čitljivost. Formatiranje, u nekim jezicima (na primer, Python) čak utiče na značenje programa.

Formatiranje i vizualni elementi programa treba da olakšaju razumevanje koda koji se čita, ali i pronalaženje potrebnog dela koda ili datoteke sa nekim delom programa.

U modernim programskim jezicima dužina reda programa nije ograničena. Ipak, predugi redovi mogu da stvaraju probleme. Formatiranje i vizualni elementi programa treba da olakšaju i proces pisanja programa.

PISANJE KOMENTARA

Čak i ako se autor pridržavao mnogih preporuka za pisanje jasnog i kvalitetnog koda, ukoliko kod nije dobro komentarisan, njegovo razumevanje može i samom autoru predstavljati teškoću već nekoliko nedelja nakon pisanja.

Komentari treba da olakšaju razumevanje koda i predstavljaju njegov svojevrsni dodatak.

Postoje alati koji olakšavaju kreiranje dokumentacije na osnovu komentara u samom kodu i delom je generišu automatski (npr. Doxygen).

Komentari ne treba da objašnjavaju ono što je očigledno: Komentari ne treba da govore kako kod radi,

već šta radi, koje su promenljive, funkcije, moduli, entiteti, klase, objekti...

MODULARNOST

Veliki program je teško ili nemoguće razmatrati ako nije podeljen na celine. Podela programa na celine (na primer, datoteke i funkcije) neophodna je za razumevanje programa i nametnula se veoma rano u istoriji programiranja.

Svi savremeni programski jezici su dizajnirani tako da je podela na manje celine ne samo moguća već tipičan način podele koji odredjuje sam stil programiranja (na primer, u objektno orijentisanim jezicima neki podaci i metode za njihovu obradu se grupišu u takozvane klase i mogu se koristiti i u drugim programima).

Podela programa na module treba da omogući razumljivost onome ko piše i onome ko čita program. Ukoliko je kod kvalitetno podeljen na celine, pojedine celine biće mogućce upotrebiti u nekom drugom kontekstu ili programu.

- ✓ **PROCESU PROGRAMIRANJA PREDHODE IZVESNE PRIPREMENE RADNJE I ONE ZAHTEVAJU:**

- POTPUNO RAZMATRANJE I PRECIZNO FORMULISANJE PROBLEMA KOJI SE REŠAVA,
- IZBOR MATEMATIČKIH METODA I POSTAVLJANJE MATEMATIČKOG MODELA,
- IZBOR I ANALIZA NUMERIČKIH POSTUPAKA SA TAČKE GLEDIŠTA EFIKASNOG REŠAVANJA DATOG PROBLEMA NA RAČUNARU.

PROGRAMIRANJE je proces zadavanja skupa naredbi u nekom programskom jeziku kako bi se izvršila neka aktivnost, odnosno, rešio određeni problem primenom računara.

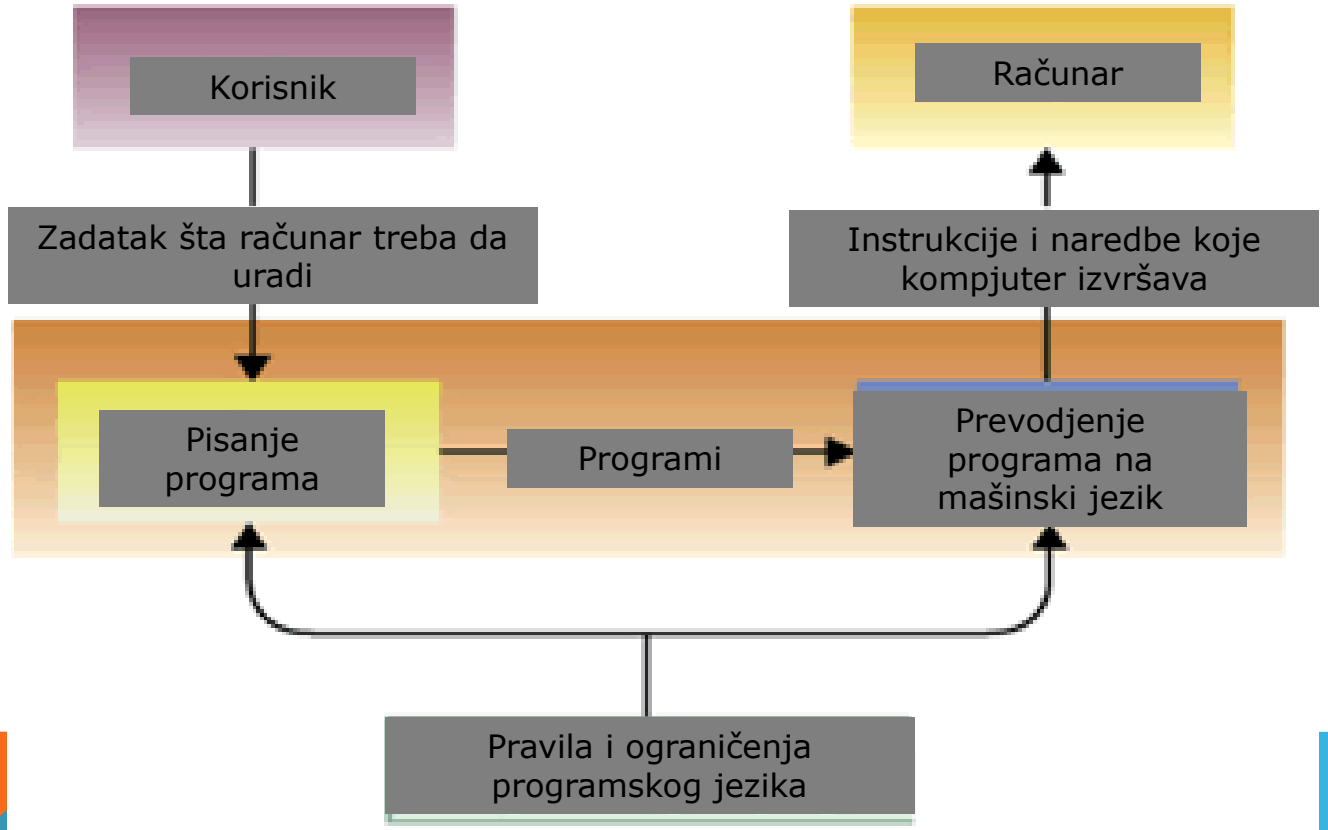
PROCES PROGRAMIRANJA OBUHVATA SLEDEĆE FAZE:

IDENTIFIKACIJA PROBLEMA	**Svaki problem može imati mnogo različitih rešenja, ali sva ona moraju imati nešto zajedničko. Znači, ovde pokušavamo da uradimo tačno ono šta se od našeg programa zahteva da uradi.**
PROJEKTOVANJE I RAZRADA ALGORITAMA	**Ako je potrebno, ili to zahteva upotreba određenog programskog jezika, vršimo projektovanje rešenja na određenoj naučnoj platformi i opisujemo ga izradom algoritma.**
DIZAJNIRANJE REŠENJA	**Jednom kada definišemo stvari potrebne za rešavanje problema, i specificiramo koju će formu zauzeti rešenje problema, potrebno je izaći na kraj sa pitanjem kako uvesti takvu specifikaciju u program na osnovu kojeg će se odraditi zadatak.**
PISANJE PROGRAMA	**Programiranje je zadatak kojim ćemo predstaviti naše dizajnirano rešenje računaru, učeći ga da po našem putu izvrši rešavanje problema** **Obično špostoje tri nivoa pisanja programa:** **n Kodiranje – (izbor EDITORA)** **n Kompilacija** **n Debugging**
TESTIRANJE PROGRAMA	**Testiranje programa je ispitivanje da li program radi ono šta smo predvideli da radi. Ovo je neizostavan proces, jer iako je kompajler proverio da li je program korektno napisan, on ne može proveriti da li ono šta smo napisali aktuelno rešava naš problem.**
IZRADA DOKUMENTACIJE	**Izrada dokumentacije je takođe neophodna jer ona spada u sisteme kvalitetnog pisanja softvera.**

Ciklični proces programiranja koji se obnavlja sve dok se ne postigne zadovoljavajuće rešenje

Šta se ovde dogadja ????

PROGRAMSKI TOK:



ELEMENTI I GREŠKE

Osnovni elementi o kojima treba voditi računa u toku programiranja su:

- brzina izvršavanja programa,
- greške u programiranju,
- ekonomično korišćenje memorijskog prostora,
- brzina programiranja.

Ovi elementi su nažalost dobrim delom protivrečni, pa se često moraju činiti kompromisi.

Greške u programiranju.

U procesu programiranja treba voditi računa o raznim izvorima grešaka koje mogu onemogućiti dobijanje zadovoljavajućih rezultata. Izvori grešaka mogu biti:

greške u postavci problema,
greške u matematičkom modelu,
nisu uzete u obzir sve moguće kombinacije ulaznih podataka

} **Čije su ovo greške ???**

JEZICI ???

Jezik:

- sistem izražavanja misli koji ima određena glasovna i gramatička pravila i služi kao glavno sredstvo za sporazumevanje među ljudima;
- način sporazumevanja uopšte.
- skup znakova, dogovora i pravila koji se koriste za predavanje, saopštavanje informacija, komunikaciju itd.

JEZICI - PODELA

- *Prirodni jezik* - jezik čija se pravila zasnivaju na svakodnevnom korišćenju bez njihove eksplicitne definicije

- *Veštački jezik* - jezik čija su pravila eksplicitno utvrđena pre njegovog korišćenja.

  - *Programski jezik* - Veštački jezik koji se koristi za pripremu programa za računar

  - *Programski jezik* - Jezik za izradu programa, sastavljen og simbola koje računar može da prevede u direktne radnje

    Algoritamski jezik - Veštački jezik namenjen opisu algoritama

KARAKTERISTIKE PROGRAMSKIH JEZIKA

Skup pravila kojima se definišu pravilne konstrukcije u nekom jeziku pa i programskom jeziku, nazivaju se gramatika toga jezika.

Svaku nauku o jeziku karakteriše:

- sintaksa (od grčkog *syntexis* - sastavljanje), nauka o sklopu jezičkih konstrukcija, obuhvata strukturu jezika i pravila za korišćenje jezika.
- struktura podataka (Programski jezik je projektovan da radi sa strukturom podataka koja može imati i vertikalnu i horizontalnu dimenziju).
- semantika (od grčkog *sematikas* - značajni), nauka o značenju (smislu) tih konstrukcija, i
- vezano vreme (bindig time).

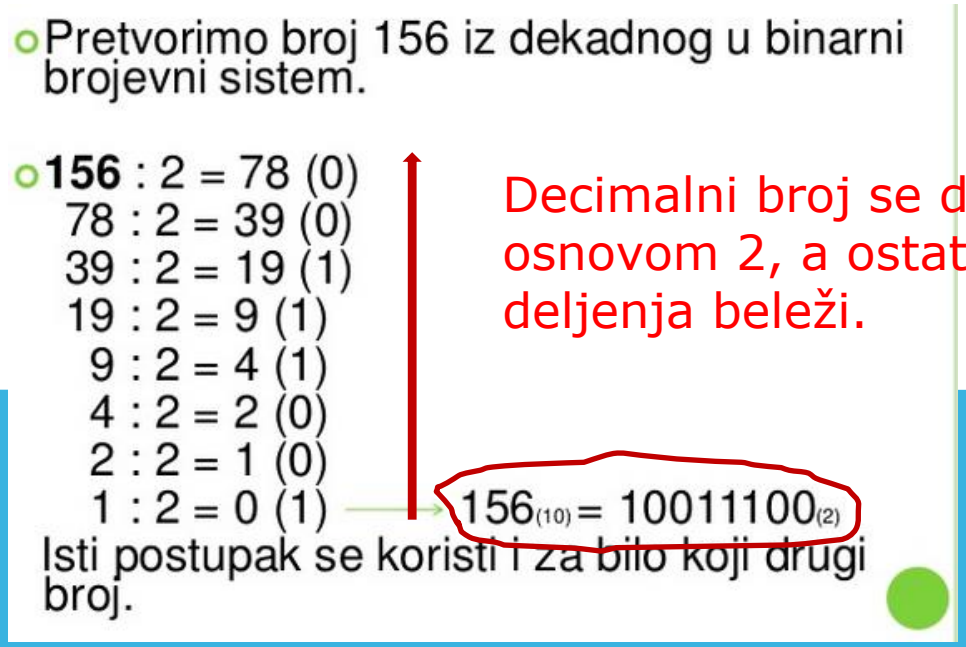
STRUKTURA PROGRAMSKOG JEZIKA

ŠTA PRE SVEGA TREBA ZNATI ???

Binarni brojni sistem (0 1) koristi BINARNI BCD kod (8 4 2 1) ili binarno kodira decimalni broj (izvorno engleski Binary-Coded Decimal) u svrhu kodiranja - odnosno prestavljanja decimalnih brojeva u binarnom obliku. Svaki decimalni broj je predstavljen sa fiksnim brojem binarnih brojeva. BCD kod se može razviti u 4 bitnom ili 8 bitnom obliku (u odnosu na procesor). Prednosti BCDa jesu veća preciznost u računanju naročito u zaokruživanju decimala, i lake čitljivosti .

Svaki decimalni broj ima jedinstveni binarni uzorak:

Decimali broj ←	BCD			
	8	4	2	1
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1



ŠTA PRE SVEGA TREBA ZNATI ???

Po težinskim delovima cifra binarnig koda se može označiti kao: $\ldots 2^4 \; 2^3 \; 2^2 \; 2^1 \; 2^0$

PRIMERI

- $1101_{(2)} = 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 + 1 \times 2^3 = 13_{(10)}$

$1001_{(2)} = 1 \times 2^0 + 0 \times 2 + 0 \times 4 + 1 \times 8 = 9_{(10)}$

$10111_{(2)} = 1 \times 2^0 + 1 \times 2 + 1 \times 4 + 0 \times 8 + 1 \times 16 = 23_{(10)}$

Isti postupak koristi se i za ostale brojeve.

ŠTA PRE SVEGA TREBA ZNATI ???

Oktalni brojni sistem je sistem zapisa brojeva sa bazom osam (oktalno kodirani decimalni broj), **danas** je redje **primenjivan u informatici i računarstvu. Koristi cifre u opsegu [0 1 2 3 4 5 6 7]. Prvih nekoliko prirodnih brojeva (skup N_0) ovog sistema upoređenih sa decimalnim glasi:**

Oktalan broj	0	1	2	3	4	5	6	7	**10**	11	12	13	14	15	16	17	**20**	21	...
Decimalan broj	0	1	2	3	4	5	6	7	**8**	9	10	11	12	13	14	15	**16**	17	...

Slede primeri konverzija:

$(11010)_2 = (011\ \ 010)_2 = (3\ 2)_8 = (32)_8$

$(1,1101)_2 = (001,\ 110\ 100)_2 = (1,\ 6\ 4)_8 = (1,64)_8$

$(254)_8 = (2\ 5\ 4)_8 = (010\ 101\ 100)_2 = (010101100)_2$

$(36,4)_8 = (3\ 6,\ 4)_8 = (011\ 110,\ 100)_2 = (11110,1)_2$

Šta se ovde dopisuje ???

Prevodjenje iz dekadnog u oktalni brojni sistem vršimo na isti način kao kod binarnog, samo deljenjem sa 8 !!!

ŠTA PRE SVEGA TREBA ZNATI ???

Heksadecimalni brojni sistem: U matematici i informatici se često koristi heksadecimalni brojevni sistem. To je brojni sistem u osnovi 16, odnosno sistem sa 16 različitih cifara. Obično se koristi [0 1 2 3 4 5 6 7 8 9 10] arapskih cifarai dodaju se slova A – B - C- D – E – F, odnosno oznake za 11 12 13 14 15 16.

Heksadecimalni sistem je pogodno koristiti u računarima pošto je pretvaranje između binarnog i heksadecimalnog sistema jednostavno. Tako svaka četiri bita mogu da se napišu kao jedna cifra heksadecimalnog sistema, što znači da se jedan bajt može napisati kao dve cifre u heksadecimalnom sistemu.

Konverzija brojeva u heksadecimalnom kodu [uredi]

Primer: Broj A2F konvertujte u odgovarajući broj dekadnog brojnog sistema i obrnuto. Rešenje:

$$A2F(16) = A \times 16^2 + 2 \times 16^1 + F \times 16^0 = 10 \times 256 + 2 \times 16 + 15 \times 1 = 2607$$

$$A2F_{(16)} = 2607_{(10)}$$

ŠTA PRE SVEGA TREBA ZNATI ???

Dakle, cifre heksadecimalnog sistema su od 0 do F po heksadecimalnom označavanju, odnosno od 0 do 15 po dekadnom shvatanju njihove vrednosti.

Heksadecimalni sistem se koristi u računarstvu u kombinaciji sa binarnim sistemom jer se pretvaranje može lako obavljati

dekadno	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
heksadek.	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
dekadno	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
heksadek.	10	11	12	13	14	15	16	17	18	19	1A	1B	1C	1D	1E	1F
dekadno	32	33	34	35	36	37	38	39	40	41	42	43				
heksadek.	20	21	22	23	24	25	26	27	28	29	2A	2B				

TABLICA KONVERZIJE

HEX / DEC / OCT / BIN

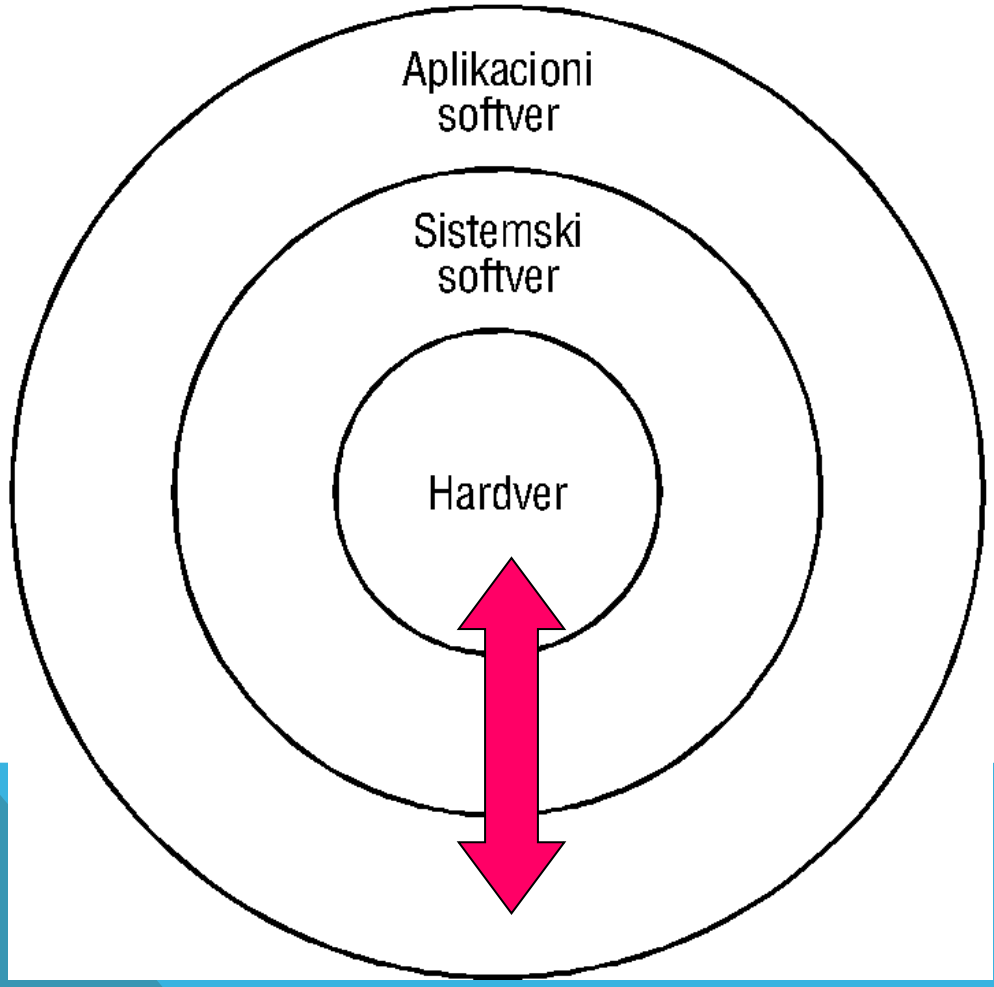
$0_{hex} = 0_{dec} = 0_{oct}$	0	0	0	0
$1_{hex} = 1_{dec} = 1_{oct}$	0	0	0	1
$2_{hex} = 2_{dec} = 2_{oct}$	0	0	1	0
$3_{hex} = 3_{dec} = 3_{oct}$	0	0	1	1
$4_{hex} = 4_{dec} = 4_{oct}$	0	1	0	0
$5_{hex} = 5_{dec} = 5_{oct}$	0	1	0	1
$6_{hex} = 6_{dec} = 6_{oct}$	0	1	1	0
$7_{hex} = 7_{dec} = 7_{oct}$	0	1	1	1
$8_{hex} = 8_{dec} = 10_{oct}$	1	0	0	0
$9_{hex} = 9_{dec} = 11_{oct}$	1	0	0	1
$A_{hex} = 10_{dec} = 12_{oct}$	1	0	1	0
$B_{hex} = 11_{dec} = 13_{oct}$	1	0	1	1
$C_{hex} = 12_{dec} = 14_{oct}$	1	1	0	0
$D_{hex} = 13_{dec} = 15_{oct}$	1	1	0	1
$E_{hex} = 14_{dec} = 16_{oct}$	1	1	1	0
$F_{hex} = 15_{dec} = 17_{oct}$	1	1	1	1

Za 8 bitni procesor:
broj n=2^8=256 simbola.
Za 16 bitni procesor:
broj n=2^{16}=65.536 simbola.

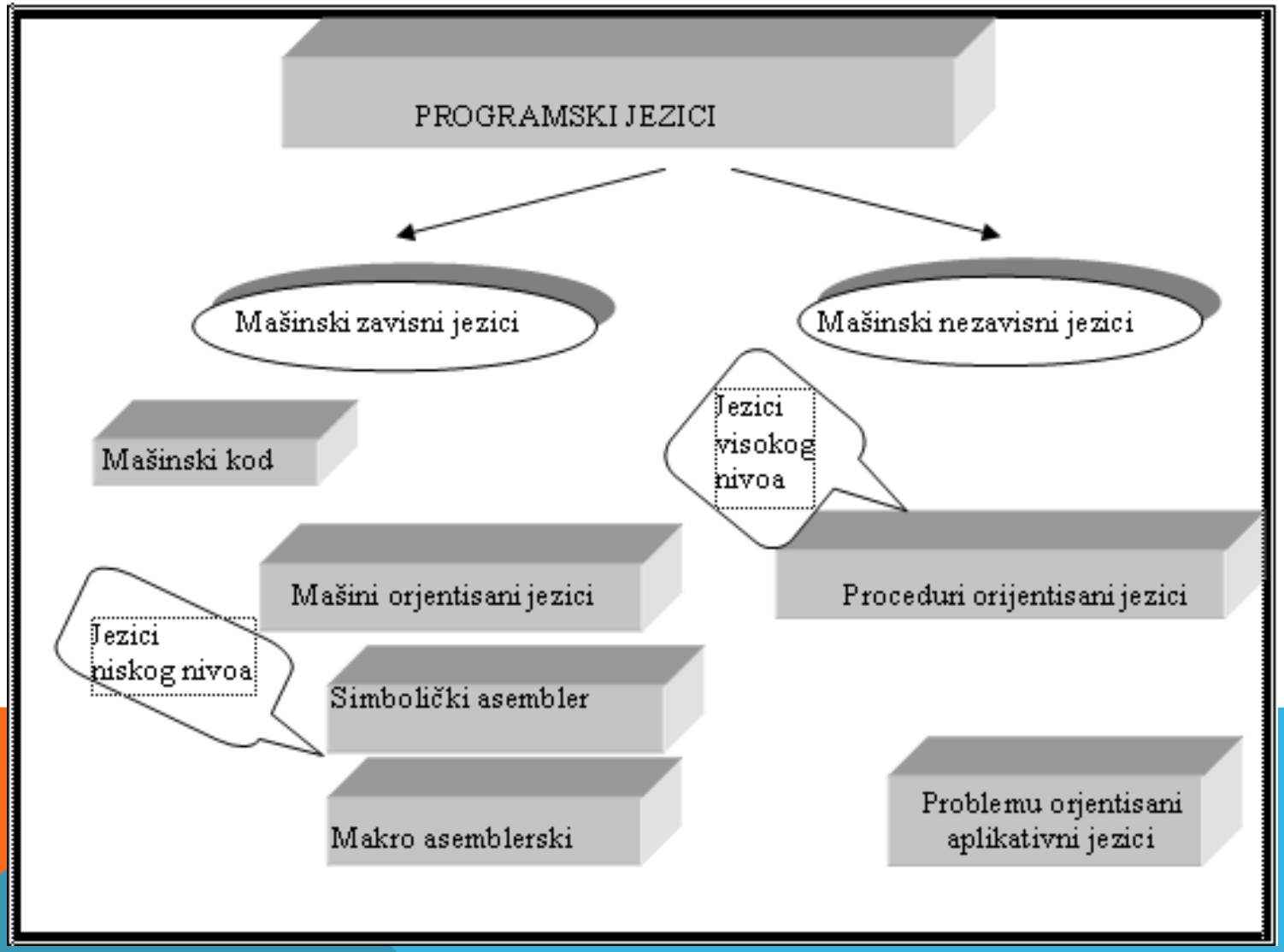
Decimal - Binary - Octal - Hex – ASCII
Konverzaciona MAPA

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	He
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F		127	01111111	177	7F

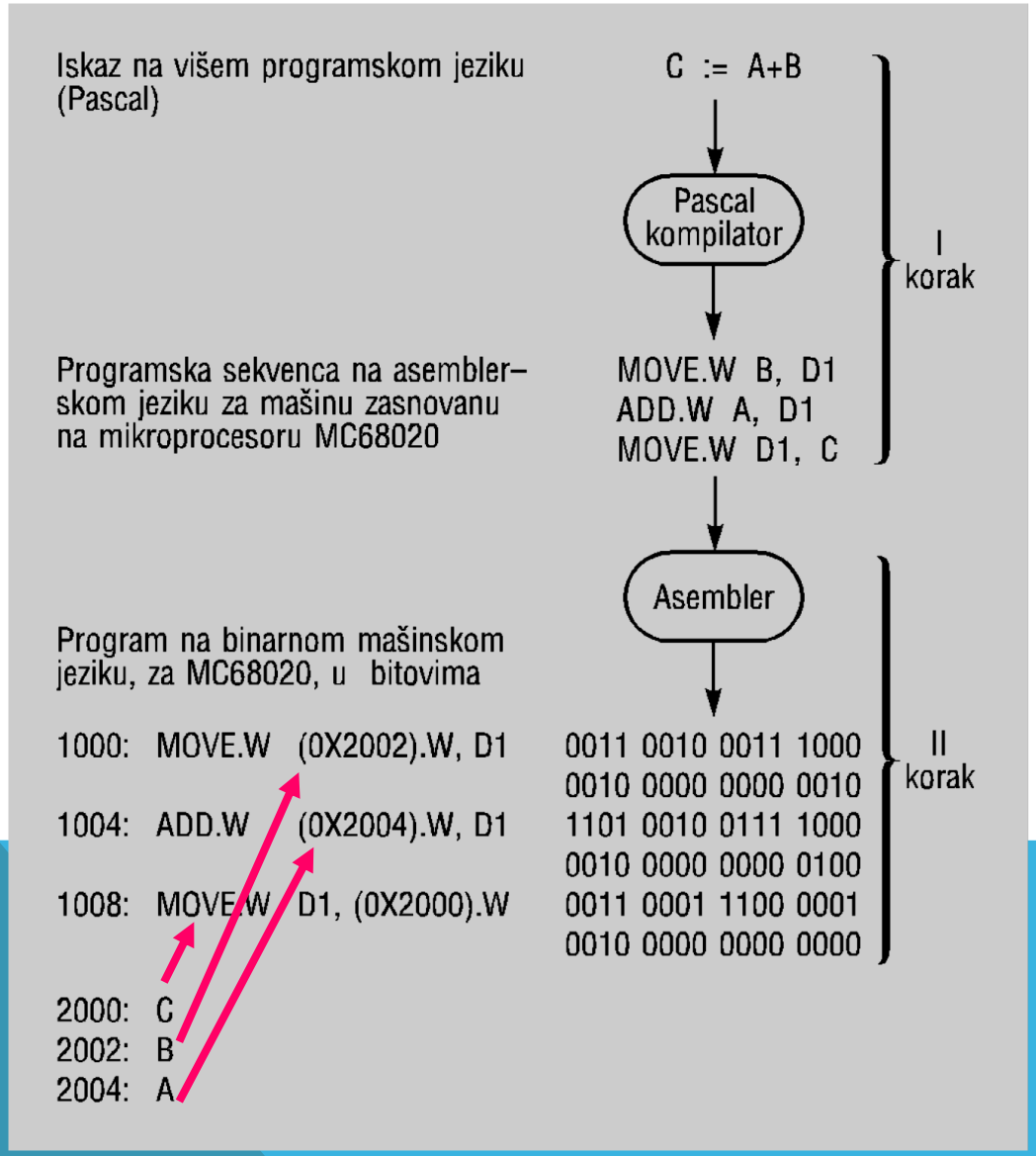
Odnos hardver, sistremski softver i aplikacioni softver kod računara



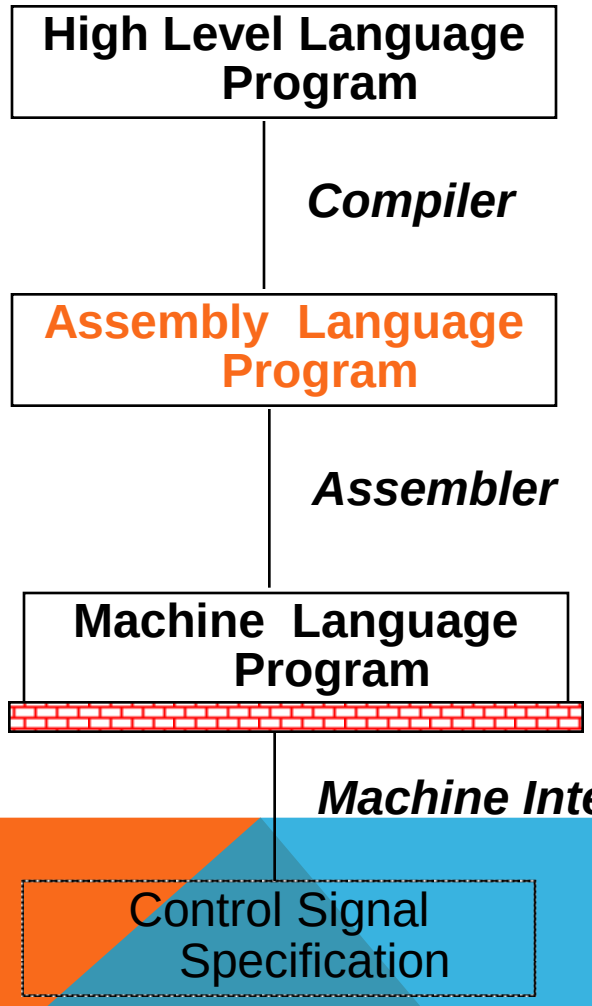
PODELA PROGRAMSKIH JEZIKA:



Pascal iskaz preveden u asemlerski jezik, a nakon toga asembliran u mašinski jezik - PROCES



Nivoi reprezentacije




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

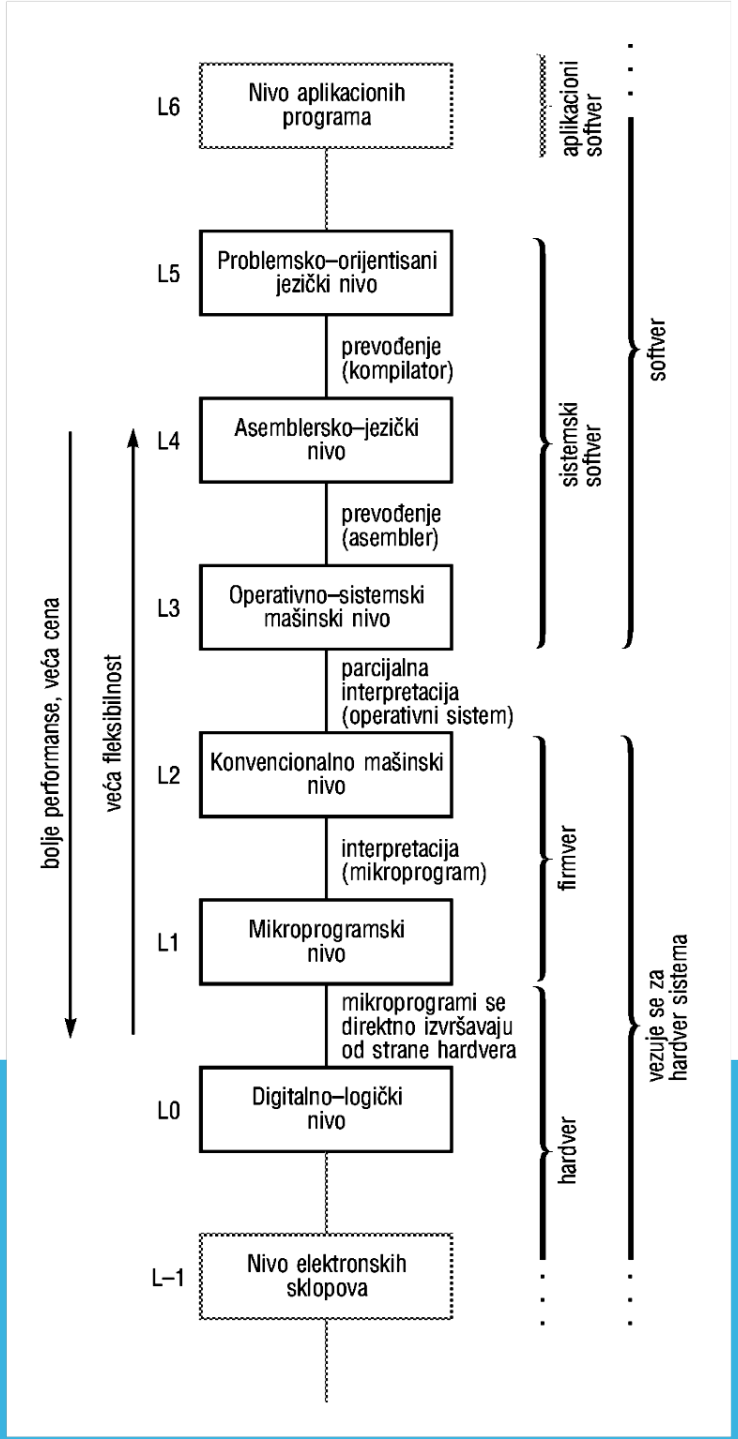
```
lw $15,   0($2)
lw $16,   4($2)
sw $16,   0($2)
sw $15,   4($2)
```

```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

ALUOP[0:3] <= InstReg[9:11] & MASK

Hijerarhijska organizacija računara - koncept nivoa

- Nivo aplikacionog programa
- Nivo višeg programskog jezika
- Nivo mašinskog kôda (asemblerski jezik)
- Upravljački nivo
- Nivo funkcionalne jedinice
- Logička kola, tranzistori i veze



Generacije računarskih jezika

Globalno razlikujemo šest klase računarskih jezika

Generacija	Opis
prva generacija	mašinski jezik
druga generacija	asemblerski jezik
treća generacija	viši programski jezici (HLL)
četvrta generacija	novi jezici 4GL
peta generacija	objektno orijentisani jezici
šesta generacija	HTML, XML i Java Web orijentisani jezici

Prva generacija – mašinski jezik

Mikroprocesor, procesor i drugi logički sklopovi računara imaju svoj vlastiti programski jezik koji se naziva mašinski jezik, a sastoji se od nizova binarnih reči koje predstavljaju: instrukcije logičkim sklopovima i podatke koje treba obraditi. Program napisan u mašinskom jeziku nazivamo izvršni program ili izvršni kod budući da ga računar može neposredno izvršiti. Mašinski jezik je određen arhitekturom racunara, a donira ga proizvođac hardvera. Izvršni program je mašinski zavistan, što znači da se kod napisan na jednom računaru ili mikroprocesoru može izvršavati jedino na računarima istog tipa.

1. svaka instrukcija, na hardverskom nivou, direktno upravlja radom mašine, tj. pojedinim gradivnim blokovima.
2. instrukcije su numeričke, predstavljene u formi binarnih oblika od 0 i 1
3. programiranje je naporno i podložno velikom broju grešaka
4. efikasnost programiranja je niska
5. programi nerazumljivi korisniku
6. direktno se pristupa resursima mašine
7. veća brzina izvršenja programa
8. efikasnije korišćenje memorije

MAŠINSKI JEZIK

Pisanje programa na mašinskom jeziku ima sledeće osobine:

- Potrebno je izvršiti detaljizaciju algoritma, tako da elementarni koraci odgovaraju pojedinim naredbama računara,
- Uvek je potrebno poznavati skup naredbi datog računara - konkretni mašinski jezik. Program napisan za jedan računar obično se može obavljati na drugom,
- Sastavljanje i pisanje programa - kodiranje kao i testiranje programa i podataka u memoriji računara zahteva detaljno poznavanje računara tako da to obavlja odgovarajući stručni kadar.

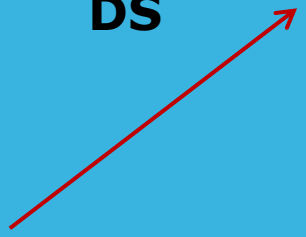
- 0001 1101 1000 0000
- 0000 0000 1111 1111
- 0001 1100 0000 0000
- 0000 0000 1111 1100
- 0001 1110 0000 0000
- 0000 0000 0111 0010

Druga generacija – asemblerski jezik

Karakteristike:

Programiranje na simboličkom jeziku ima niz nedostataka. Navedimo najvažnije: potrebno je vršiti detaljizaciju algoritma tako da elementarnim algoritamskim koracima odgovaraju dejstva simboličkih naredbi, raznovrsnost računara dovela je do raznovrsnosti simboličkih jezika, tako da svaki konkretni računar ili familija računara ima svoj poseban simbolički jezik. Program napisan na simboličkom jeziku za jedan računar ne može bez veće ili manje prerade da se izvršava na drugom.

```
0001 1101 1000 0000      LD    $R1. <A
0000 0000 1111 1111      ADD   $R1. <B
0001 1100 0000 0000      STO   $R1. <C
0000 0000 1111 1100      A     DC     5
0001 1110 0000 0000      B     DC    13
0000 0000 0111 0010      C     DS
```



DRUGA GENERACIJA

ASEMBLERSKI JEZIK

- svaka instrukcija se predstavlja mnemonikom, kao na primer ADD X
- korespondencija izmedju asemblerskih i mašinskih instrukcija je jedan-na-prema-jedan
- postoje i direktive koje nemaju izvršno dejstvo, ali programu na asemblerskom jeziku olakšavaju prevođenje, dodelu memorije i segmentaciju programa
- direktno se pristupa resursima mašine
- veća brzina izvršenja programa
- efikasnije korišćenje memorije

Treća generacija – viši programski jezici

HLL

Karakteristike:

1. Kompajler prevodi programske iskaze u odgovarajuće sekvence instrukcija na mašinskom nivou, mašinski nezavisni
2. U principu jedan iskaz na HLL-u (*High Level Language*) se prevodi u *n* (n >= 1) instrukcija na mašinskom (asemblerskom), jezik je jednostavniji, sintaksa na Engleskom jeziku
3. Efikasnost je veća, problemski orijentisani
4. Ispravljanje grešaka lakše
5. Nema direktni pristup resursima mašine
6. Neefikasno iskorišćenje memorije
7. Duži programi

VIŠI PROGRAMSKI JEZICI

Ada: Dobio je naziv po imenu Augusta Ada Byron koji se smatra prvim poznatim programerom. Razvijen od strane američke vlade kao standard viših programskih jezika koji treba da zameni COBOL i FORTRAN.

BASIC: (Beginner's all-Purpose Symbolic Instruction Code). Jednostavni proceduralni jezik za programiranje od strane korisnika.

C: Strukturirani jezik srednjeg nivoa razvijen kao deo UNIX operativnog sistema. Sličan je mašinski zavisnom asemblerskom jeziku po efikasnosti ali je mašinski nezavisan i ima sve mogućnosti viših programskih jezika.

COBOL: (COmmon Business Oriented Language) Programski jezik čija se sintaksa zasniva na engleskom jeziku. Ima veoma široku primenu u programiranju poslovnih aplikacija

FORTRAN: (FORmula TRANslation) Viši programski jezik dizajniran za naučne i inženjerske primene.

PASCAL: Dobio je naziv po Blaise Pascal. Razvijen je specijalno da uključi koncepte strukturnog programiranja i struktura podataka.

TREĆA GENERACIJA – HLL

Programiranje je znatno olakšano. Ovi jezici su bliski čovekovom pisanom jeziku i operativnoj terminologiji iz odgovarajuće oblasti,skraćeno je vreme obuke u programiranju i izbegnute su teškoće oko detalja u vezi sa programiranjem na mašinskom, odnosno simboličkom jeziku,

Ovi jezici su nezavisni od strukture samog računara. Program napisan na ovom jeziku može da se izvršava na svakom računaru koji ima prevodilac (kompajler) za ovaj jezik,

Postoji mogućnost izmene programa i iskustva između korisnika jednog problemsko - orjentisanog jezika.

ČETVRTA GENERACIJA– novi jezici

4GL

Novi tipovi računarskih jezika se karakterišu sledećim osobinama:

- Implementiraju veštačku inteligenciju (primer je LISP)
- Jezici za pristup bazama podataka (primer je SQL)
- Objektno-orijentisani jezici (primeri su C++, Java i dr.)
- Neproceduralni, deklarativni i konverzacioni jezici, programer i korisnik specificira rezultate koje želi da dobije (npr upitnim jezikom SQL) ali ne i NAČIN kako se dobijaju

Jezici **ČETVRTE GENERACIJE** - 4GL

SQL skript

```
SELECT IME, PREZIME, BR_IND
FROM studenti
WHERE BR_IND=I2017
            (SELECT BR_IND
             FROM prijave
             WHERE OCENA>5 AND
                        DATUM=”juni2018”)
```

- Nemaju efikasnost i fleksibilnost jezika treće generacije ali su jednostavni za upotrebu
- Kompromis kombinovanjem : pretraživanje podataka korišćenjem SQL-a a upravljanje programom putem proceduralnih jezika (C, PL/1,…)

Objektni jezici

- Objekat se sastoji od podataka i operacija koje mogu biti izvršene nad podacima.(primjer štedni račun i obračun interesa)
- Kod objektnih jezika objekti aktiviraju druge objekte da izvršavaju operacije nad sobom (primer pozivanje funkcije)
- Mogućnost višestrukog korišćenja jednom definisanog objekta je glavna prednost objektnog programiranja.
- Većina objektno orijentisanih paketa za programiranje obezbjeđuje GUI tj. *point and click* i *drag and drop* vizuelno sastavljanje objekata ili tzv. vizuelno programsko okruženje (Visual Basic, Visual C++ i drugi).

Web orijentisani jezici

- **HTML (Hypertext Markup Language)** je jezik za kreiranje hypertext i hypermedia dokumenata.

- **XML (eXtensibile Markup Language)** nije jezik za formatiranje Web strana već on opisuje njihov sadržaj primenom identifikacionih tagova ili kontekstualnih labela za podatke na Web strani. (primer označavanje proizvoda na zalihama na Web strani, sa tagovima *vrsta*, *cena*, *veličina* ubrzava pretragu)

- XML omogućava jednostavniji i efektivniji e- commerce, podržavajući i unapređujući elektronsku razmenu poslovnih podataka između kompanija i njihovih klijenata

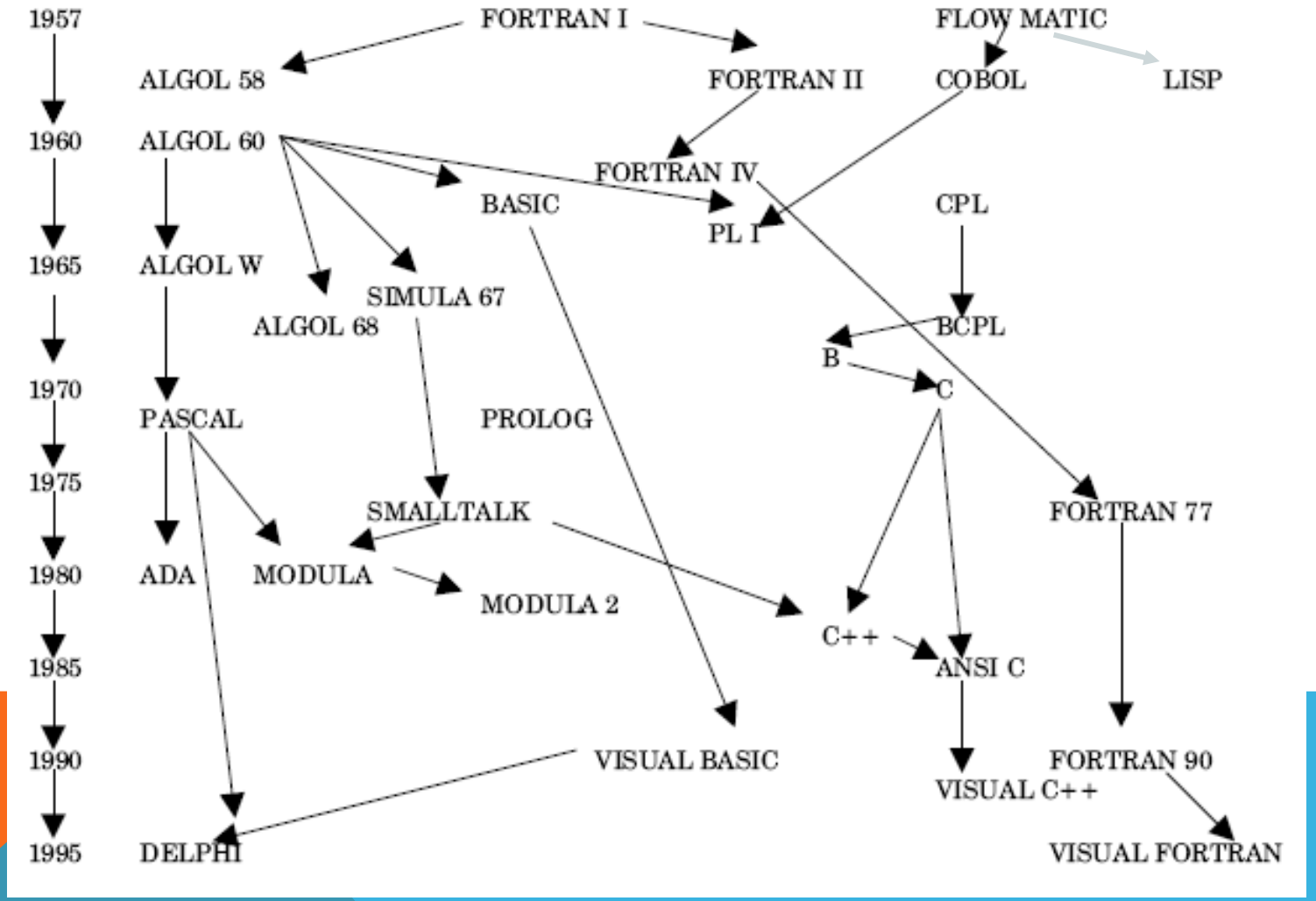
PODELA PROGRAMSKIH JEZIKA PREMA OBLASTI PRIMENE

- naučne aplikacije,
- poslovnu obradu,
- veštačku inteligenciju,
- projektovanje sistemskog softvera,
- projektovanje pomocu računara (CAD),
- upravljahje pomocu računara (CAM),
- opis hardvera računara,
- simboličko programiranje,
- linearno programiranje,
- simulaciju,
- računarske komunikacije,
- prostorno planiranje,
- stono izdavaštvo,
- obradu teksta (tekst procesori)

KARAKTERISTIKE PROGRAMSKIH JEZIKA

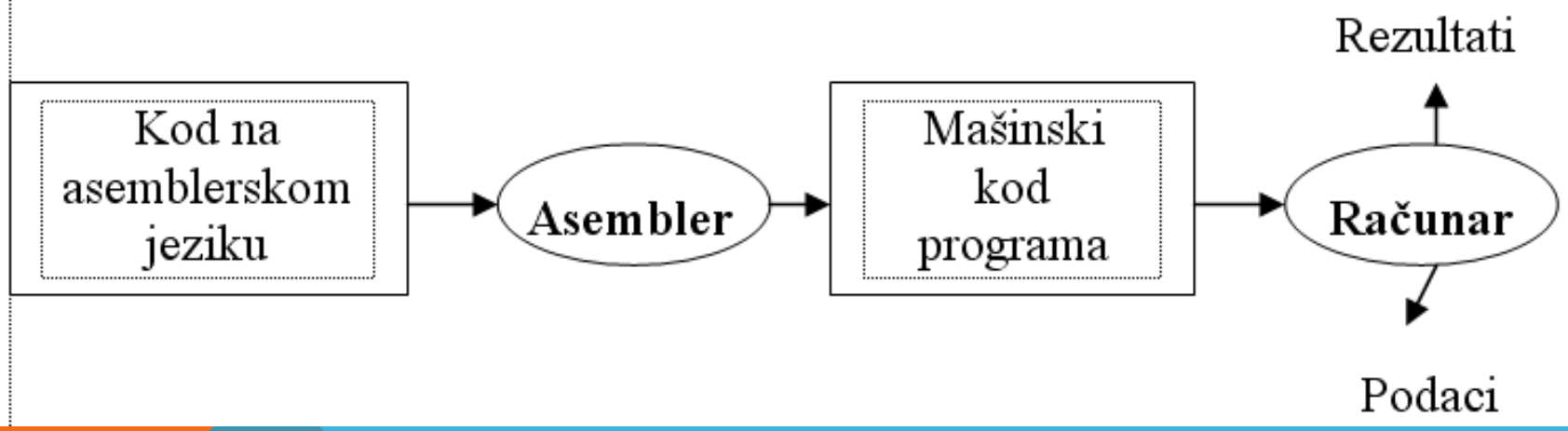
- ***Formalno definisana sintaksa programskog jezika***
-
- ***Jaki tipovi podataka***
- ***Strukturni tipovi podataka (vektori i matrice, polja)***
- ***Upravljačke strukture*** **(strukture za definisanje selekcija (if, case) i iteracija (for, while)).**
- ***Potprogrami***
-
- ***Moduli***

PRIKAZ RAZVOJA VIŠIH PROGRAMSKIH JEZIKA

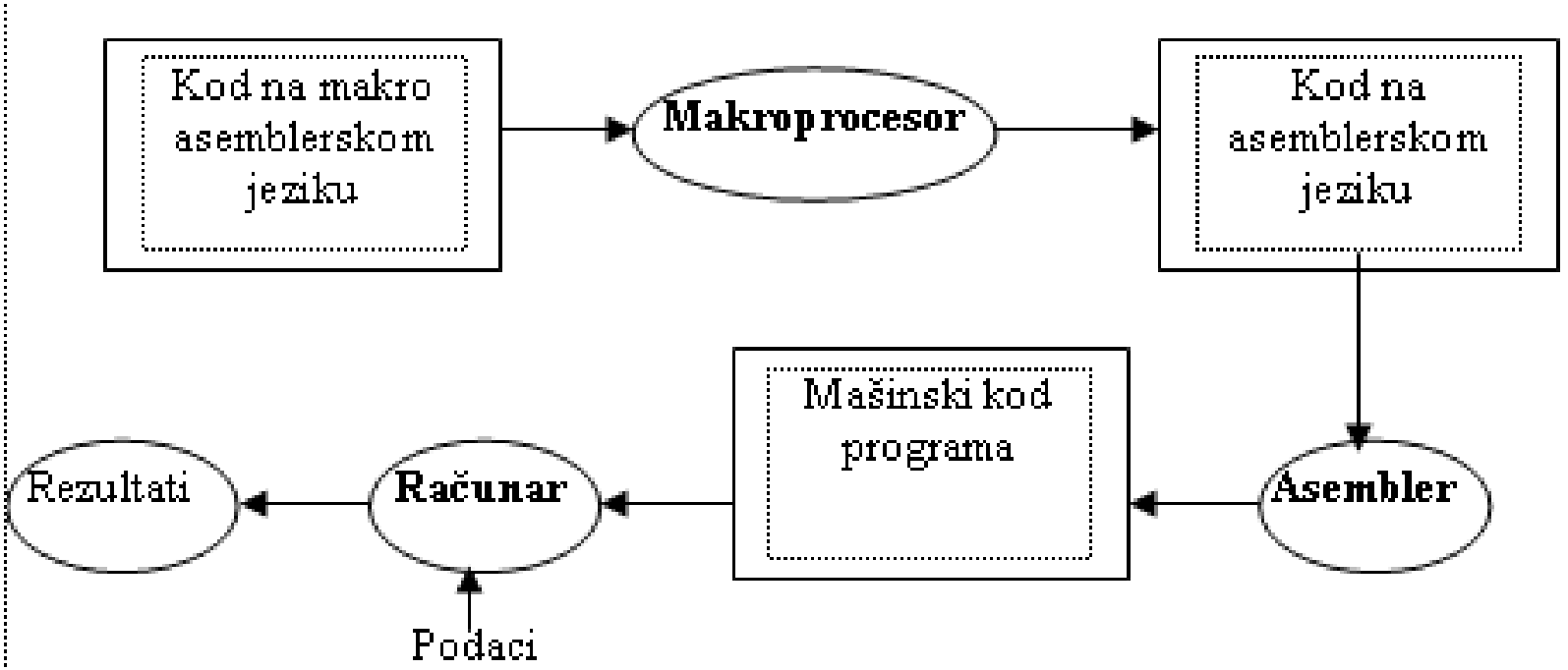


ASEMBLERI - PREVODJENJE

0001	1101	1000	0000		LD	$R1.	<A
0000	0000	1111	1111		ADD	$R1.	<B
0001	1100	0000	0000		STO	$R1.	<C
0000	0000	1111	1100		A	DC	5
0001	1110	0000	0000		B	DC	13
0000	0000	0111	0010		C	DS	

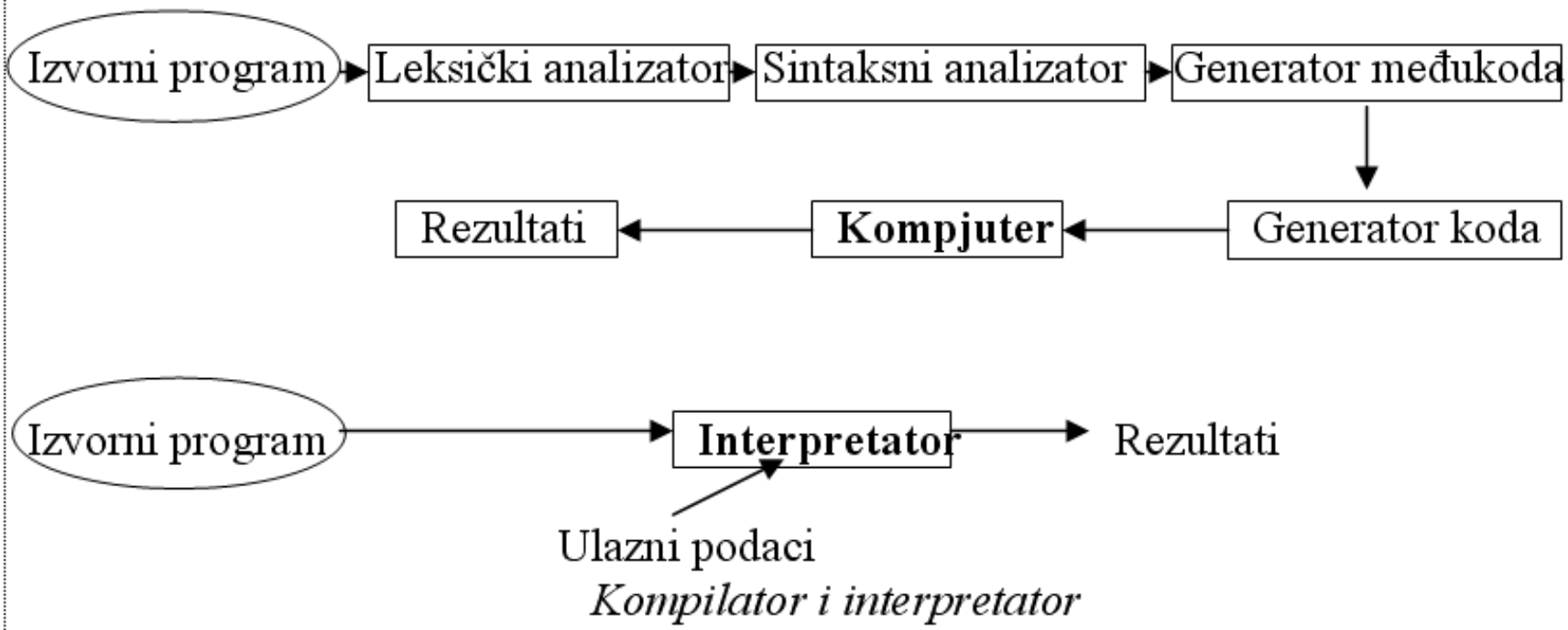


MAKRO ASEMBLERI - PREVODJENJE



U okviru naprednijih asemblerskih jezika obično postoje mogućnosti za definisanje makro naredbi, složenijih naredbi koje predstavljaju sekvence više asemblerskih naredbi. Jednom definisane makro naredbe mogu se višestruko koristiti u programu u kome su definisane.

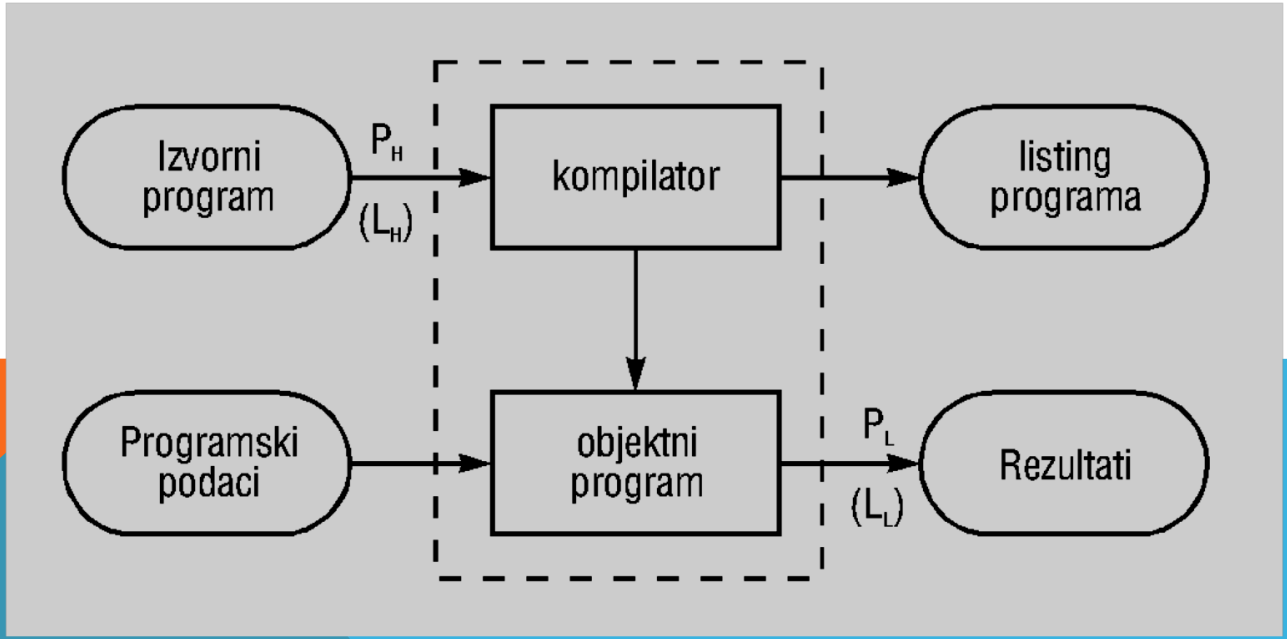
KOMPILATORSKI I INTERPRETATORSKI JEZICI
PREVODJENJE



Kompilator i interpretator

Mehanizmi za prevodjenje sa HLL-a na mašinski kôd

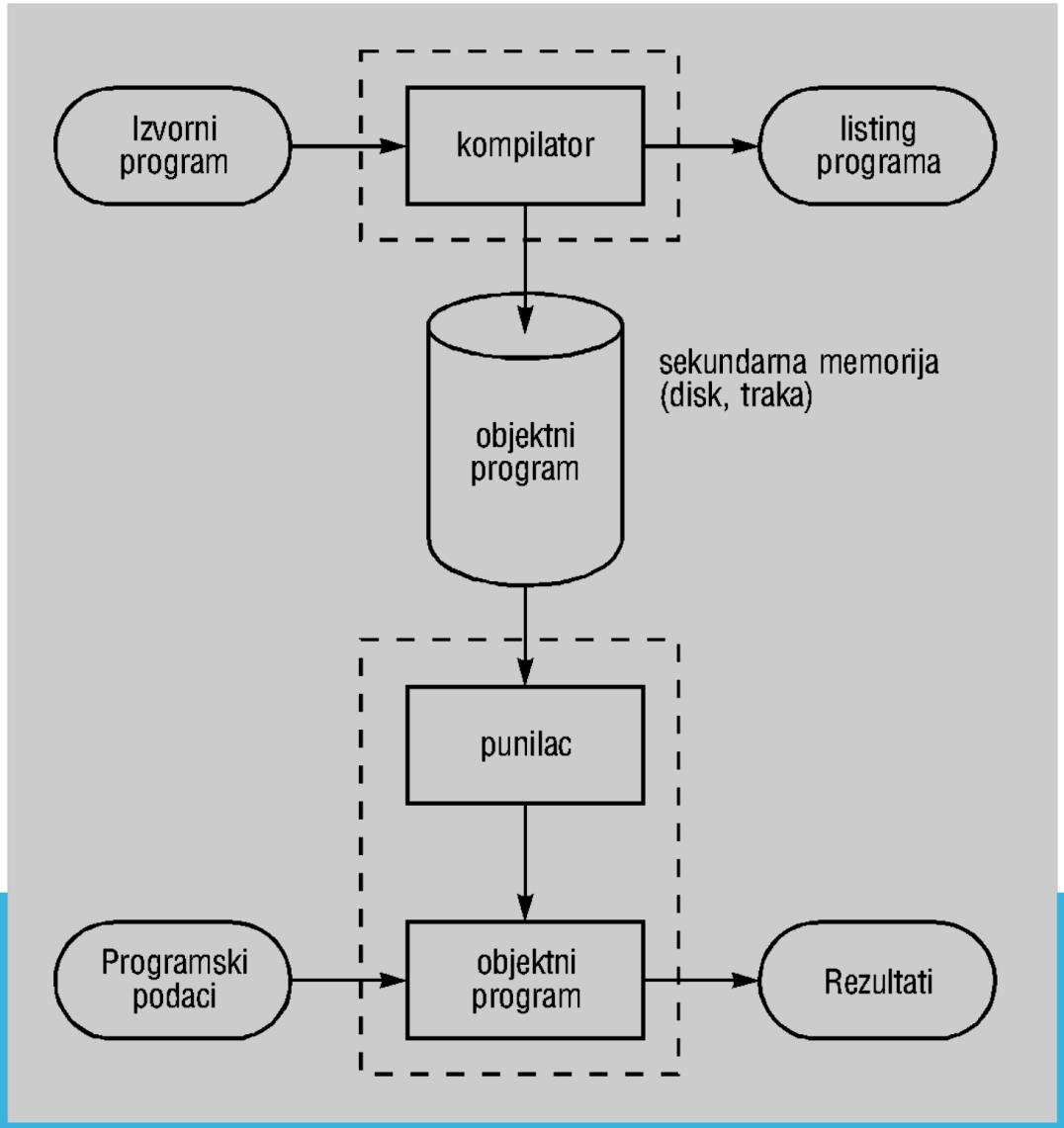
kompilacija – kompilator je program koji na svom ulazu prihvata program napisan na nekom od programskih jezika nazvan izvorni program, a na svom izlazu generiše ekvivalentan program na mašinskom kôdu nazvan objektni program, tj. program u formi mašinskog kôda koji se može direktno izvršavati od strane hardvera računara



Ciklus kompilacije

- Objektni program koji se generiše na izlazu kompilatora upisuje se u sekundarnu memoriju

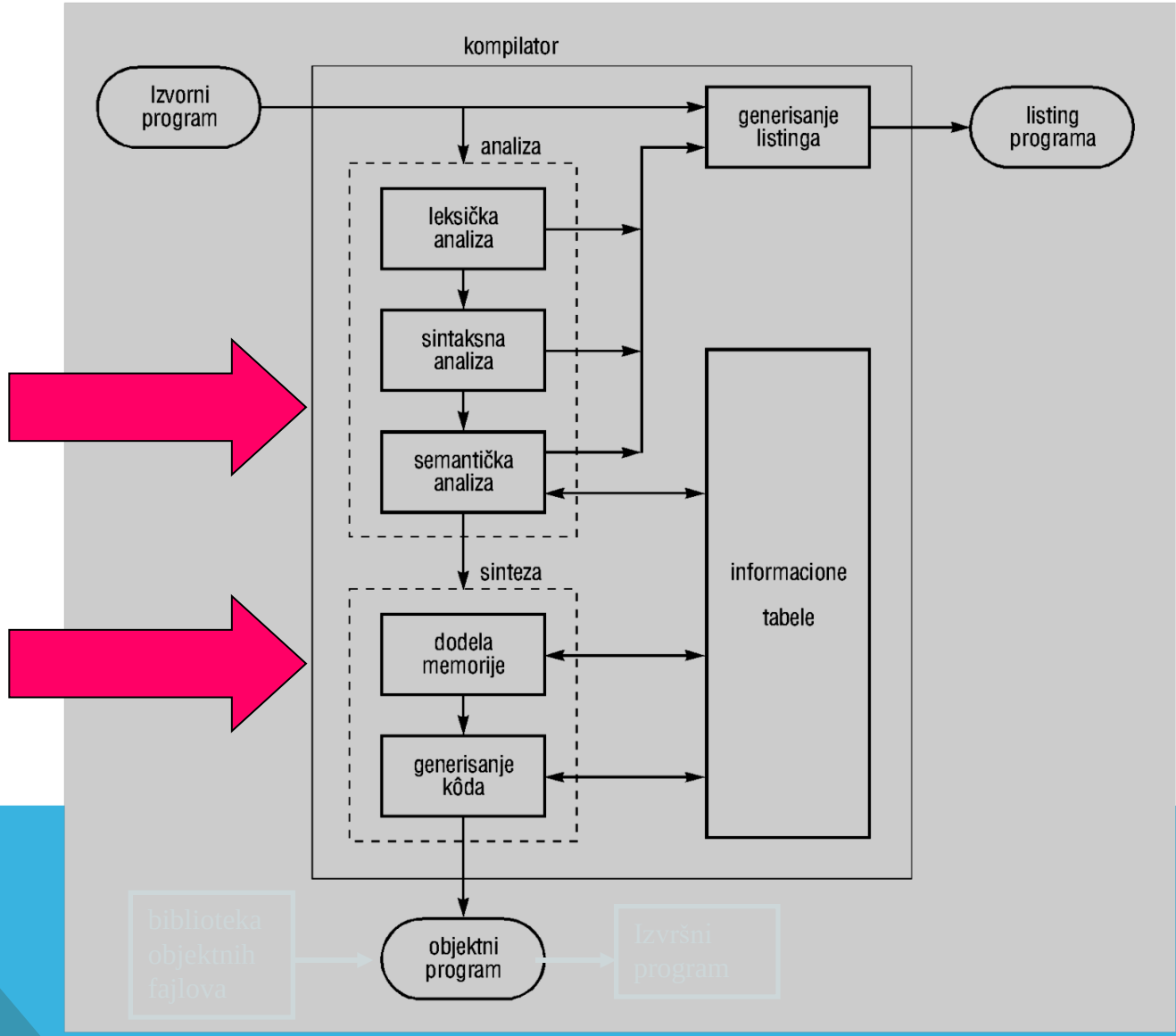
- Korisnički program poznat kao punilac (*loader*) smešta program iz sekundarne meorije u glavnu čime je objektni program spreman za izvršenje



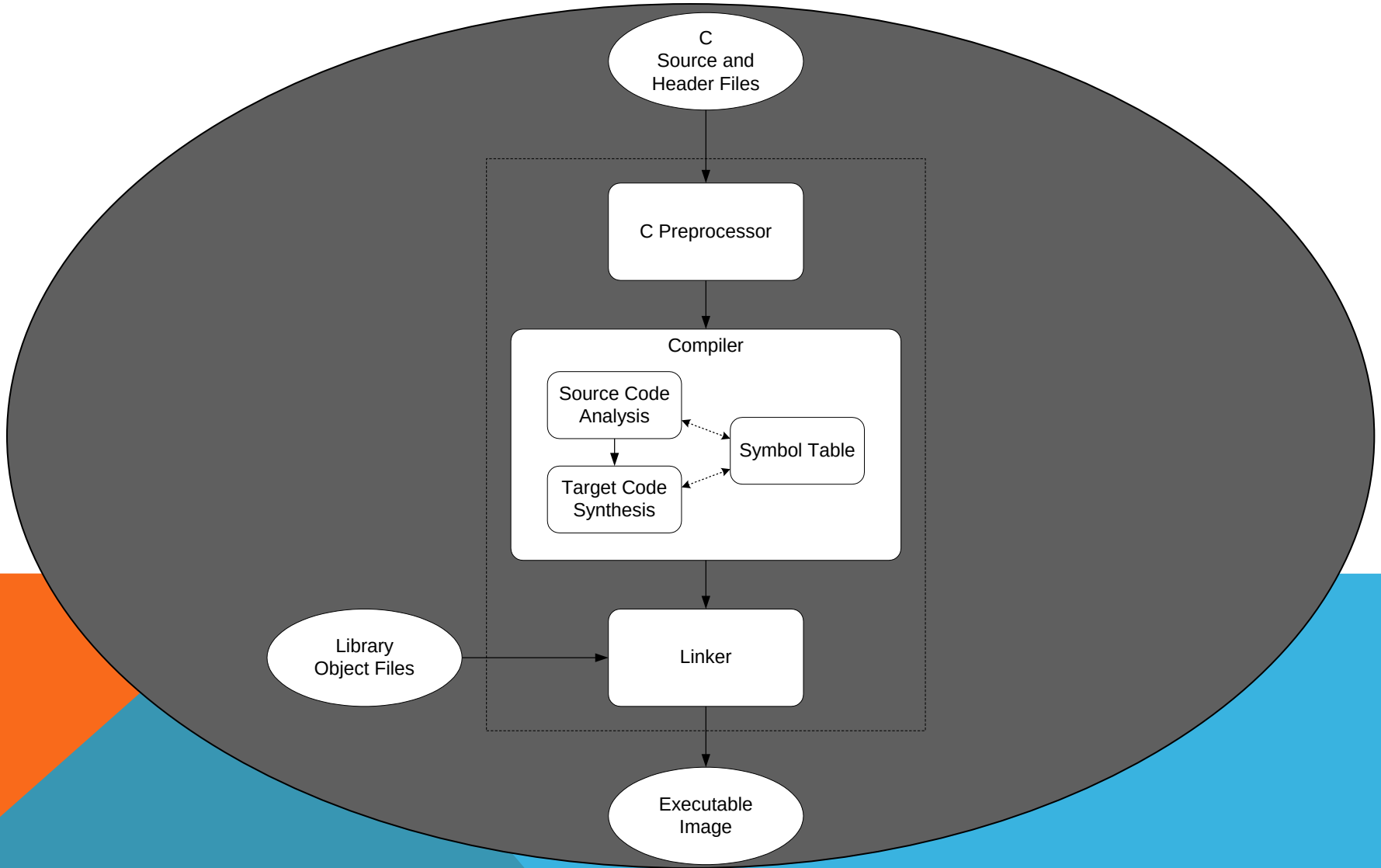
Proces kompilacije

- Proces kompilacije se može podeliti na dva glavna dela:

- analiza izvornog programa
- sinteza objektnog programa

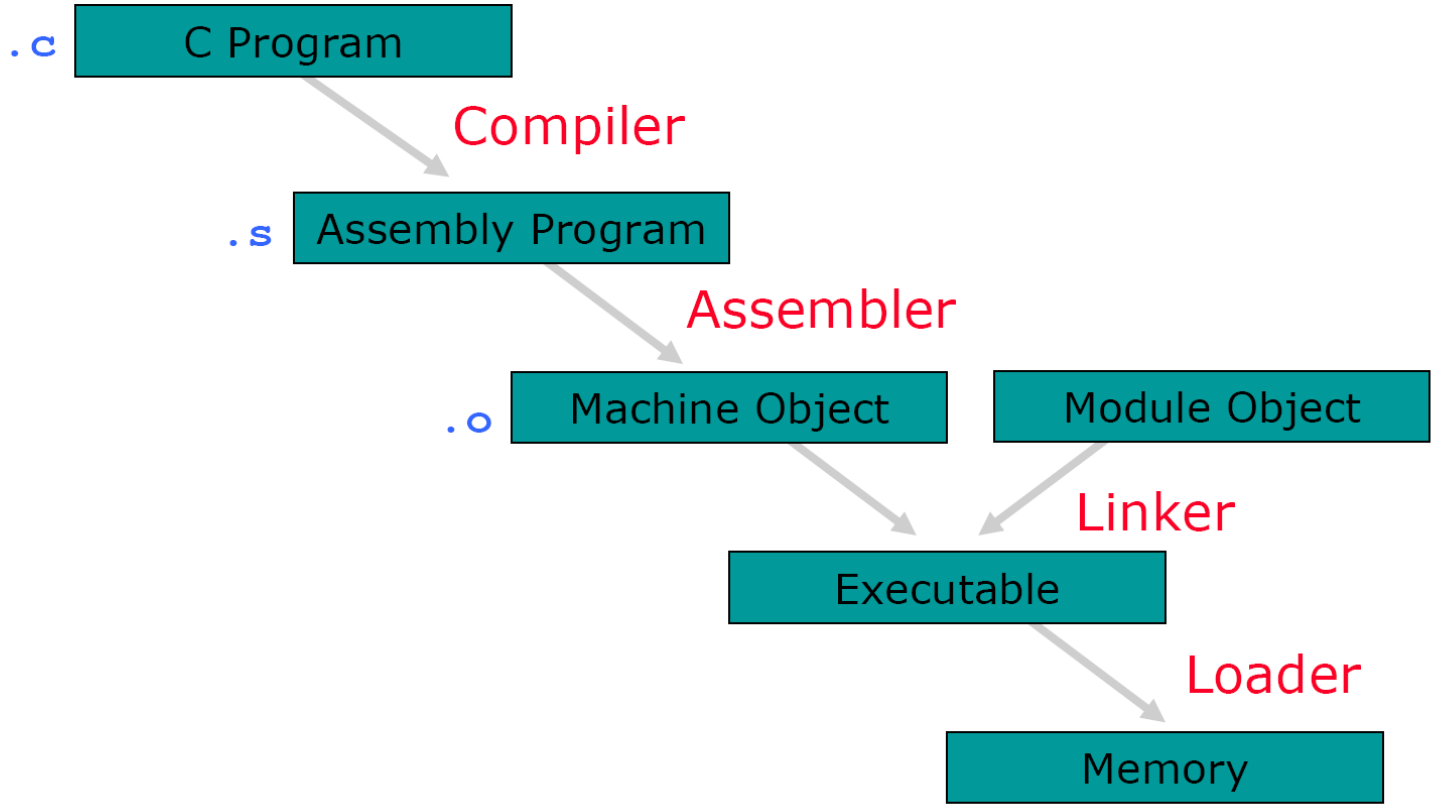


KOMPILACIJA C JEZIKA



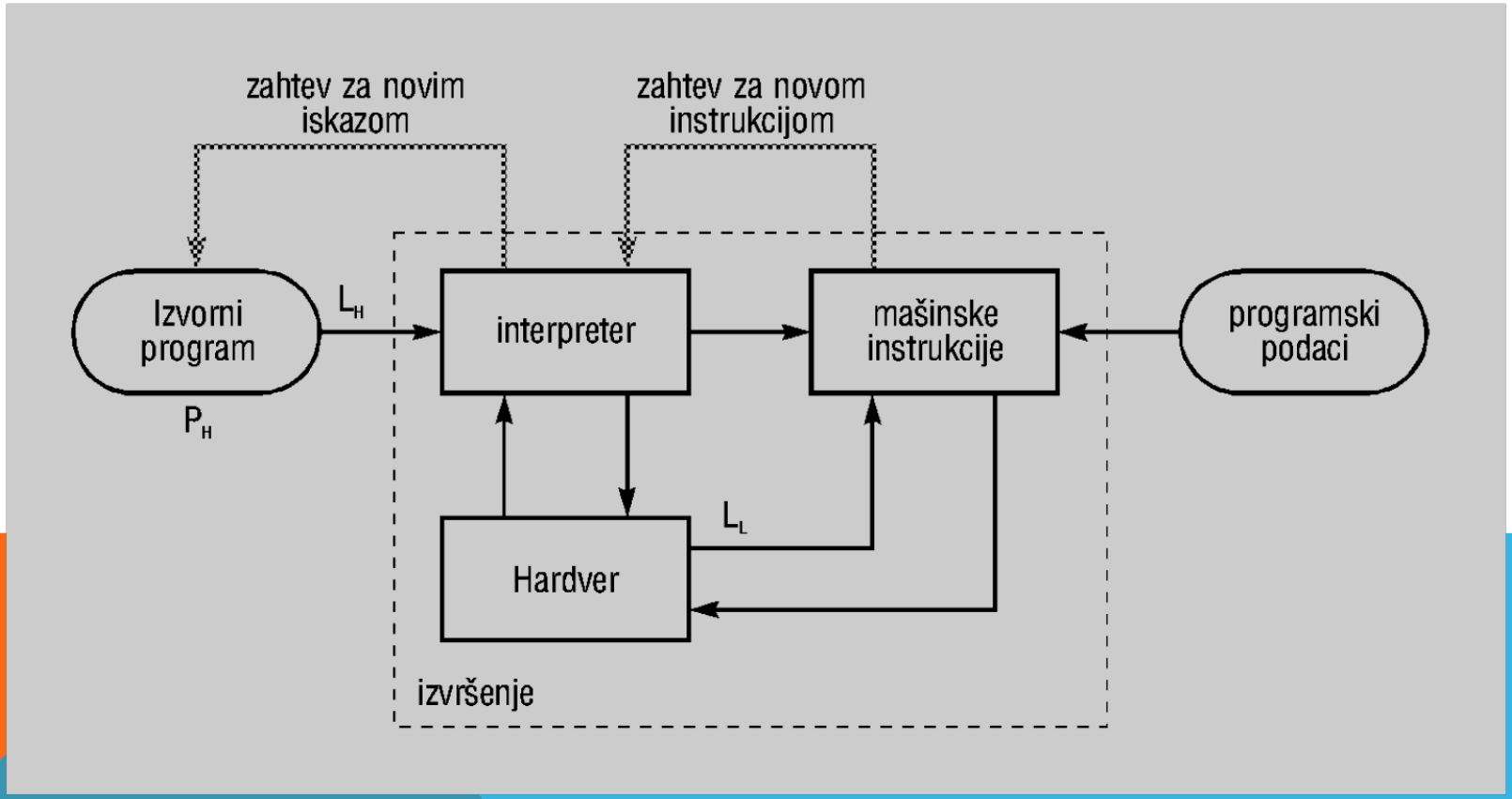
HIJERARHIJA PROCESA

- High-level → Assembly → Machine



PREVODJENJE - Interpretatori

Interpreter uzima jednu instrukciju iz programa P_H, analizira je i uslovljava da se sa istim efektom izvrši niz instrukcija sa nivoa L_L. Ovaj proces se nastavlja sve dok se ne izvrši kompletan program



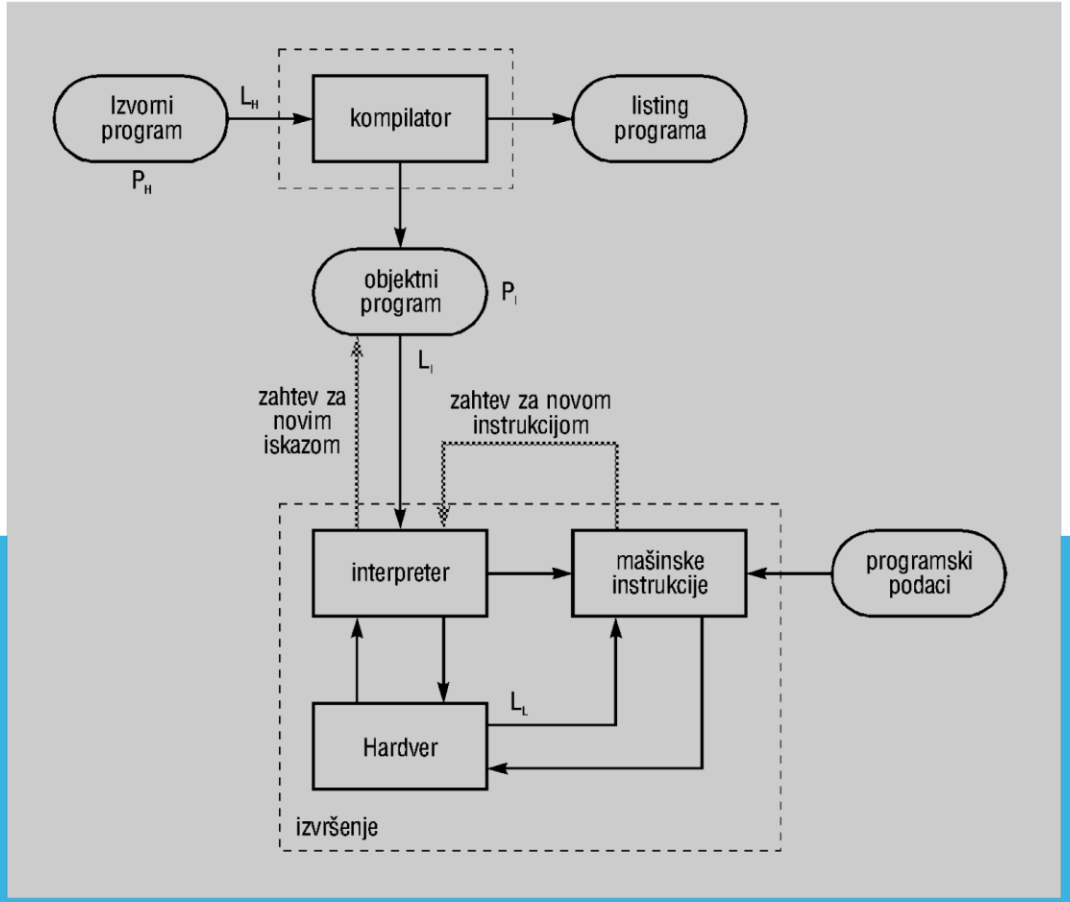
Interpretacija - prednosti i nedostaci

- **Prednost** interpeterskog mehanizma konverzije je ta što je interpreter relativno mali i što se u odnosu na kompilator lakše imlementira na nivou mašine.
- **Nedostatak** je taj što je izvršenje izvornih programa postupkom interpetacije sporije u odnosu na kompilovane programe
- Razlika u brzini izvršenja je reda 10

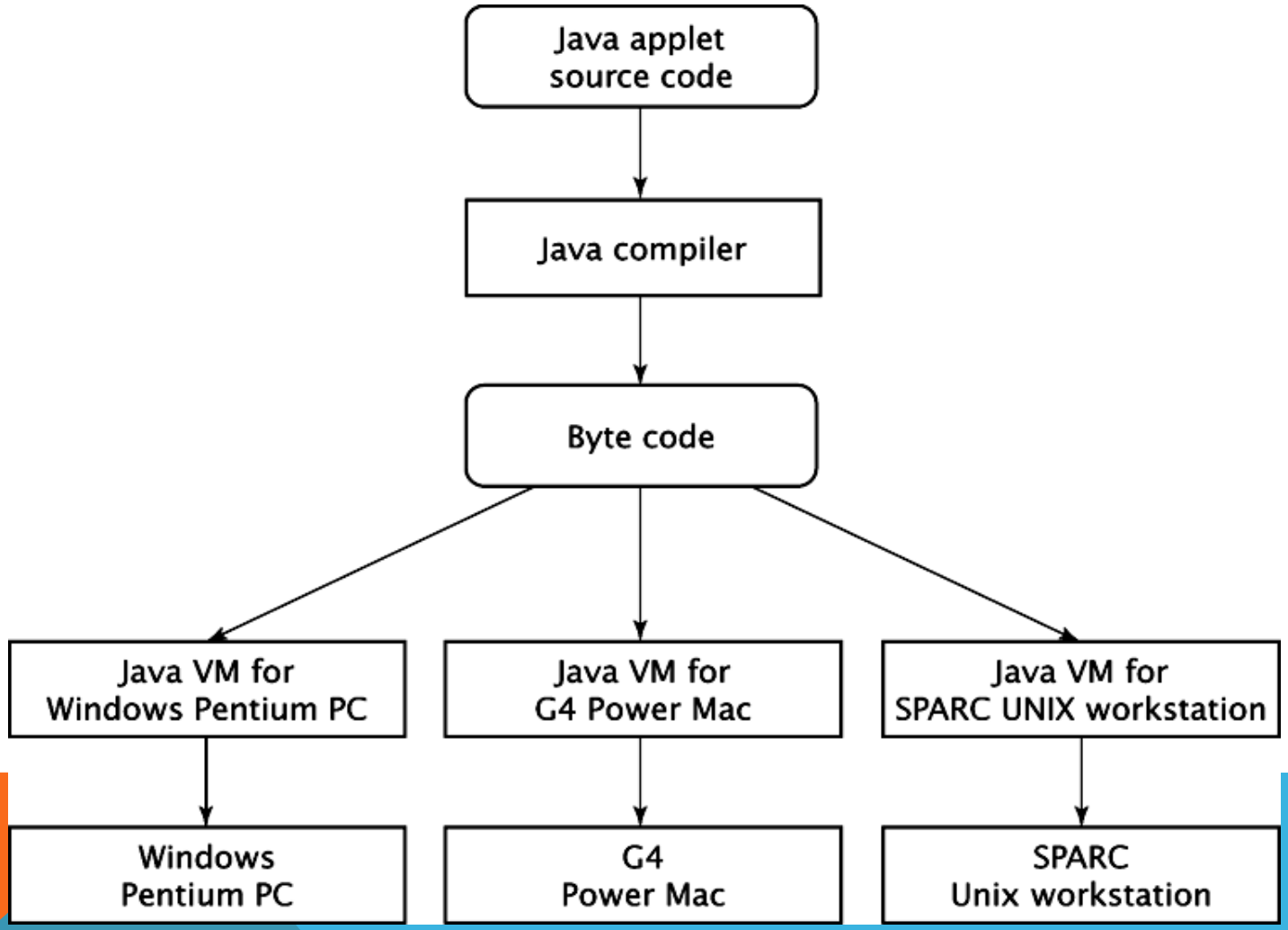
Kombinovanje interpetacije i kompilacije

Kombinovani mehanizam kompilacija-interpretacija koristi među-jezik L_I, koji se nalazi između nivoa L_L i L_H. Program P_H, napisan na jeziku L_H, kompajlira se u program P_I na jeziku L_I. Program P_I se zatim intepretira na nivou L_L pomoću interpretera

Prednost ove metode ogleda se u boljoj prenosivosti programa, kao i u tome što se zadržavaju ostale dobre osobine tehnika kompilacije i interpretacije

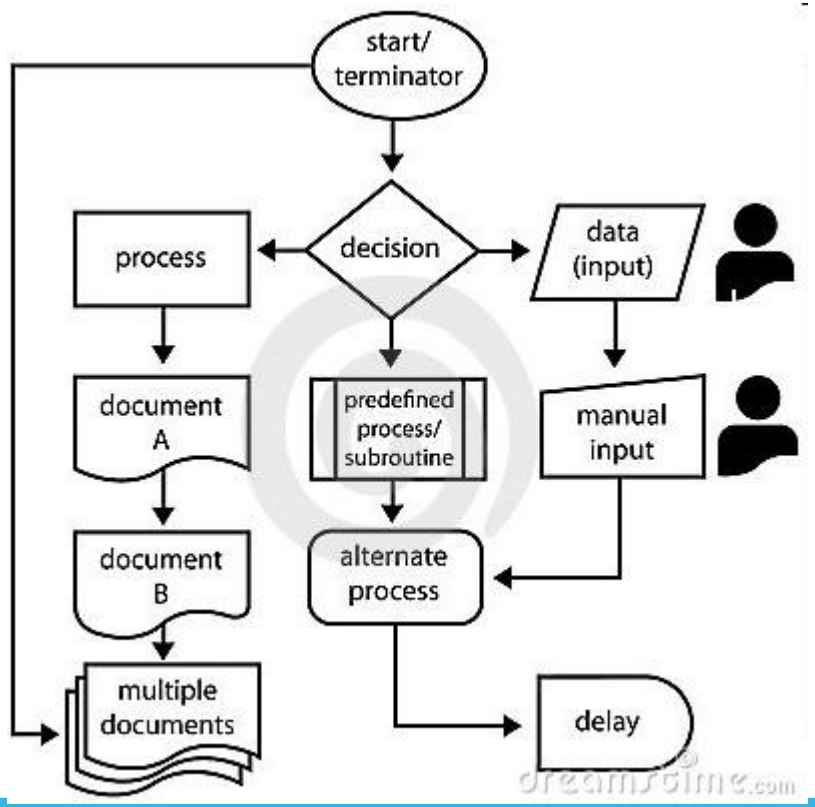


Proces kompilacije za Java aplete

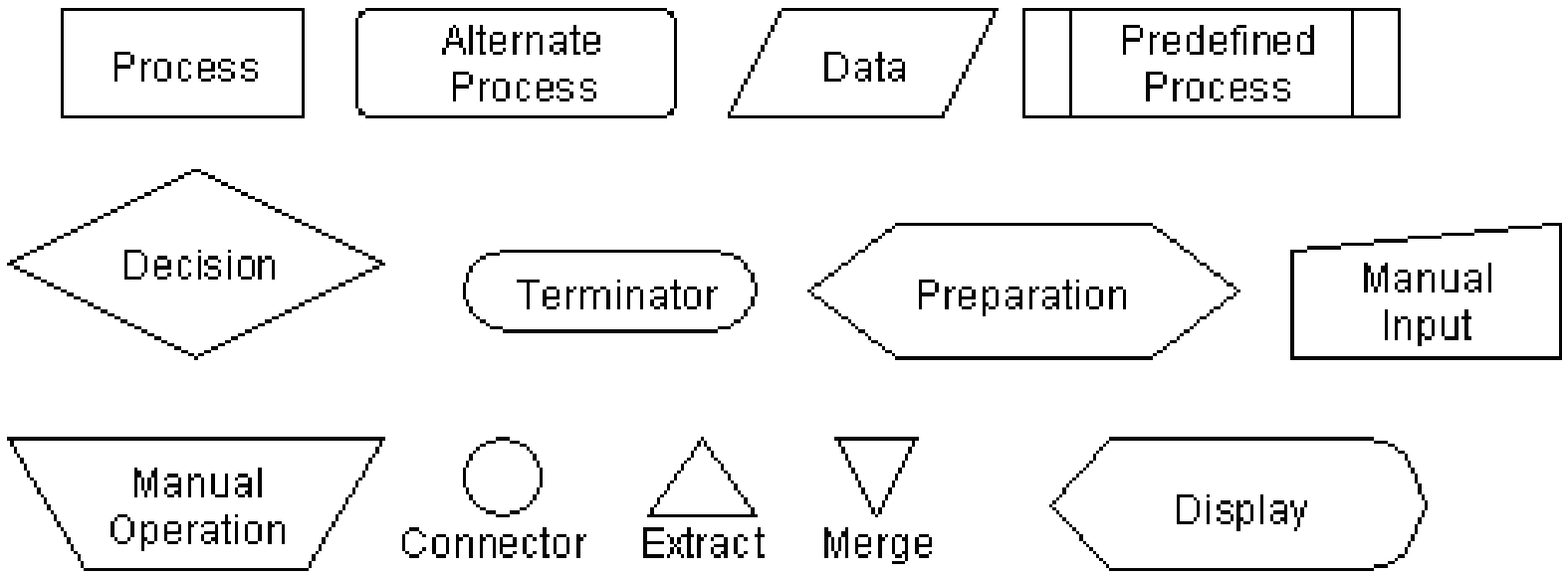


SISTEMSKI DIJAGRAMI

Glavna svrha sistemskih dijagrama je definisanje ulaza i izlaza programa, odgovarajućih ulazno-izlaznih medijuma, kao i ukazivanje na funkciju programa, bez ulaženja u problem kako to računar radi. Sistemski dijagram se odnosi na jedan prolaz (run) programa kroz računar. Pod tim se podrazumeva vremenski period u kome računar kontroliše jedan program i njime upravlja da prihvati određenu strukturu ulaznih podataka i da na svom izlazu da određenu strukturu izlaznih podataka. Kod crtanja sistemskih dijagrama koriste se standardni grafički simboli



KOMPONENTE BLOKOVA ZA PRIKAZ TOKA ALGORITAMA



Components of a Block Diagram or Flow Chart

ALGORITAM

- **Algoritam predstavlja skup akcija sa definisanim redosledom njihovog obavljanja, koji primijenjen na polazni skup podataka, dovodi do traženih rezultata**

- **U procesu programiranja, skup akcija definisan je mogućnostima računara, odnosno naredbama programskog jezika koji se koristi, dok se redosled izvršavanja akcija zadaje pomoću algoritamskih (programskih) struktura**

ELEMENTARNE ALGORITAMSKE STRUKTURE

Postoje tri elementarne algoritamske strukture:

Linijska - sve akcije se izvršavaju tačno jednom u redosledu u kome su navedene

Razgranata (SELEKCIJA) - omogućuje da se od više grupa akcija, koje se nalaze u različitim granama razgranate strukture, izabere ona koja će se izvršiti jednom, dok se sve ostale grupe akcija neće izvršitini nijednom

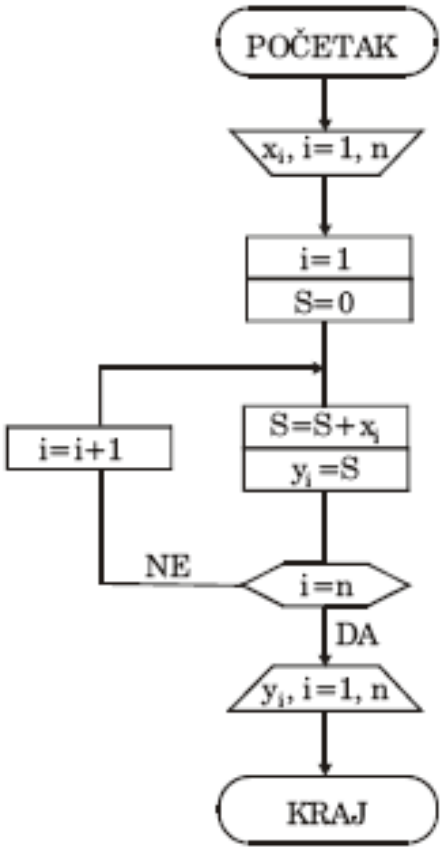
Ciklična (ITERACIJA) - skup akcija može se izvršiti više puta

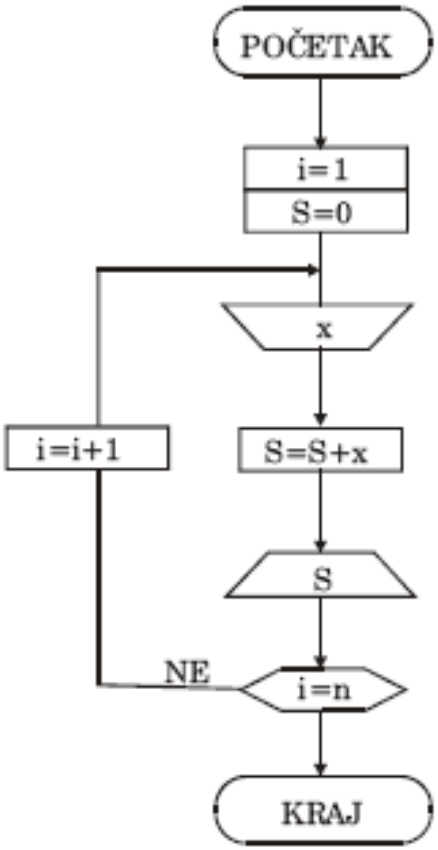
Algoritamsko rešenje bilo kog problema može se uvek zapisati korišćenjem samo ove tri strukture

PRIMER:

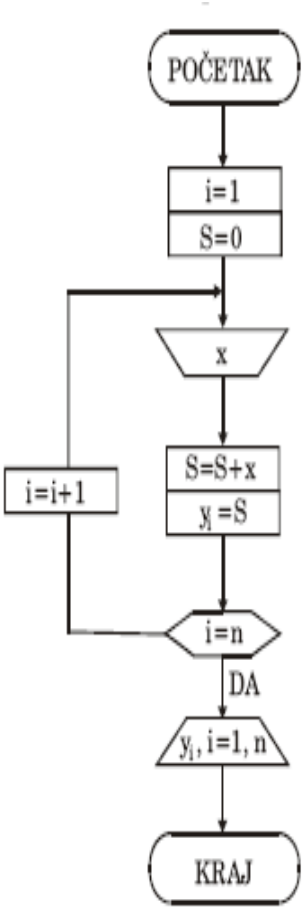
Dat je niz $\bar{x} = \{x_i\}$ od n elementa. Treba izračunati i štampati sve sume definisane relacijom

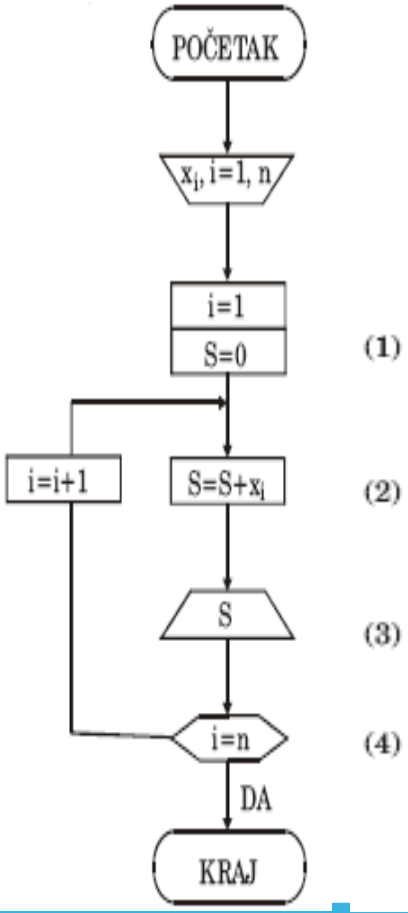
$$S_j = \sum_{i-1}^{j} x_i, \quad j = 1,\ldots,n$$

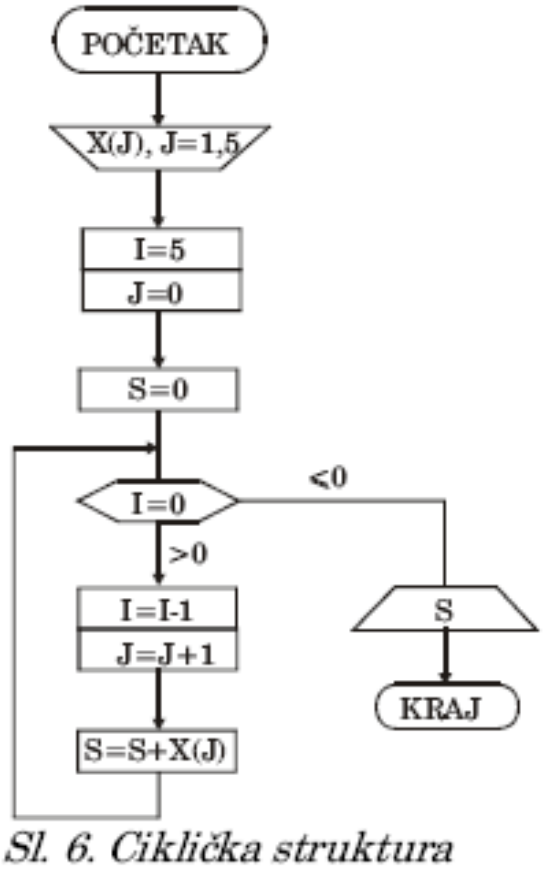




ILI







Sl. 6. Ciklička struktura

UOPŠTE O C JEZIKU

Programski jezik C je viši programski jezik opšte namene. Razvio ga je Dennis Ritchie sedamdestih godina prošlog stoleća u Bell Telephone Laboratories, Inc.

Opis jezika dat je u knjizi Brian W. Kernighan, Dennis M. Ritchie:

The C Programming Language, Prentice-Hall, 1978.

Tokom sedamdesetih i osamdesetih godina jezik se brzo širio pa je American National Standard Institute (ANSI) pristupio njegovoj standardizaciji koja je dovršena 1989. godine.

HVALA NA PAŽNJI - NASTAVAK SLEDI