3. PREZENTACIJA PODATAKA

3.1. Klasifikacija podataka

Podaci koji se predstavljaju u računaru mogu se podeliti u jednu od sledeće tri kategorije:

1. **Podaci generisani od strane korisnika** - eksplicitno se specifisiraju od strane korisnika. Struktura i karakteristike ovih podataka određeni su mogućnostima korišćenog programskog jezika.
2. **Sistemski podaci** - podaci koji se implicitno generišu od strane računarskog sistema kada se program izvršava.
3. **Instrukcije** - program koji se izvršava se može razmatrati kao kompozicija podataka, koja ima svoju sopstvenu strukturu i posebne karakteristike.

Pre nego što se upustimo u analizu korisnički definisanih tipova podataka analizirajmo prvo tipove podataka koji se najčešće sreću u svakodnevnom životu. I pored toga što se ovi podaci mogu predstaviti u različitim oblicima i formama, oni se grubo mogu podeliti na brojeve i simbole.

Brojevi se mogu zapisati na različite načine pri čemu se najčešće sreću sledeći oblici:

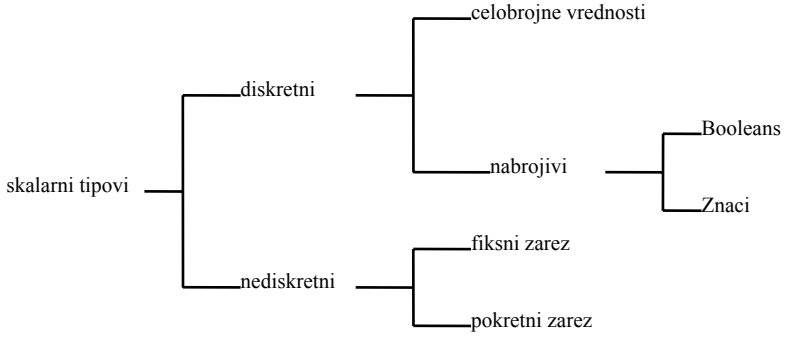
- **Prirodni brojevi** (*cardinals* ili *natural numbers*), kao što su 1; 2; 3; ...; 12148; ...
- **Celobrojne vrednosti** (*integers*) - skup celih brojeva, kako pozitivnih, tako i negativnih uključujući i nulu.
  ...; -268, ..., -1; 0; +1; ...; 9163; ...
- **Racionalni** - skup celobrojnih i razlomaka, $-1\frac{1}{4}$; 22/7
- **Realni brojevi** (*floating point numbers*) - ovaj skup sadrži one brojeve koji se ne mogu izraziti kao razlomci.
  $\sqrt{3}$; $\log_2(13)$; $\sin(\pi/4)$
- **Kompleksni brojevi** - čini ih realni i imaginarni deo.
  $4+3i$; $2-3i\sqrt{3}$

Simboli se koriste za simboličku predstavu podataka, kao što su pisma, crteži. Kada su u pitanju simboli kažemo da postoji veći izbor simbola u poređenju sa brojevima, pre svega u pogledu njihove prezentacije (grčka slova, latinica, ćirilica i dr.). Primer:

a; A; b; B; ...; Zdravo; Rim; α, β, ..., Π, Γ, ...

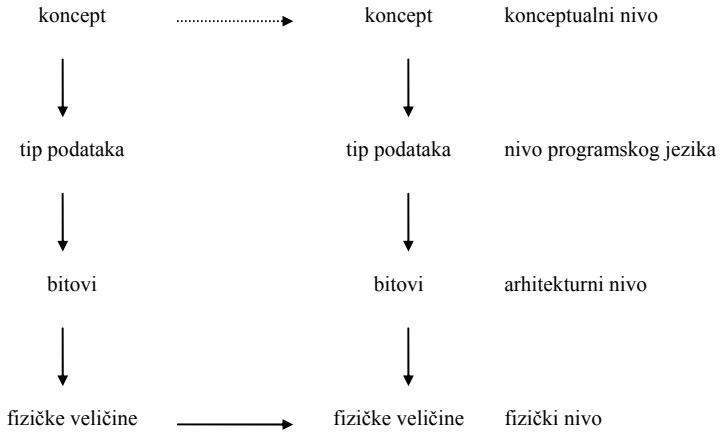
3.2. Skalarni tipovi podataka

Sa arhitekturne tačke gledišta, skalarni tipovi podataka se mogu klasifikovati saglasno slici 3.1.



Sl. 3.1. Klasifikacija skalarnih tipova podataka.

Ove tipove podataka koristi većina programskih jezika i oni predstavljaju slike ili aproksimacije sveta koji nas okružuje, na primer, koncepta razlomka 1/3. U realnim uslovima, konceptima se mora manipulisati. Ipak, kako se ovo ne može direktno postići u računaru, mora se koristiti, kao što je prikazano na slici 3.2, nekoliko međukoraka. Da bi se tipovi podataka prihvatili od strane računara, oni se moraju predstaviti u logičkim celinama, obično bitovima. Imajući u vidu da je bit apstraktna notacija on se mora konvertovati u fizičku veličinu kao što je električni potencijal ili magnetno polje, tako da se sa njim u električnom smislu može manipulisati.



Sl. 3.2. Prezentacija i konverzija tipova podataka.

Svaki tip podatka ima dve osobine:
- skup vrednosti koje podatak može uzimati (domen),
- operacije koje su dozvoljene na tom podatku.

Domen se karakteriše u sledeća dva aspekta:
- opseg (*range*) - broj vrednosti koje podatak može uzimati,
- preciznost - razlika između sukcesivnih vrednosti podataka.

3.2.1. Celobrojne vrednosti

Celobrojne vrednosti se koriste za egzaktno računanje, kao upravljačke promenljive iteracije, kao indeksi u poljima itd. Kod računarskih sistema je njihov opseg ograničen, zbog njihove dužine, tj. broj bitova koji se koristi za njihovu prezentaciju je ograničen izvedbom samog računara.

Celobrojne vrednosti se mogu predstavljati u obliku dvoičnog komplementa, jediničnog komplementa i u prezentaciji znak moduo. Operacije nad celobrojnim vrednostima daju tačan rezultat, a za njih važe aritmetička pravila. Indikaciju je potrebno dati kada je rezultat van opsega koji se može predstaviti (premašaj tj. overflow). Operatori iz viših programskih, jezika koji se primenjuju nad celobrojnim vrednostima, su:

**	eksponent
*, mod, /, div	množenje, moduo i deljenje
+ i -	znakovni operatori (unarni operatori)
+ i -	sabiranje i oduzimanje (binarni operatori)
=, $, <, >, <=, >=	realacioni operatori

I kod nabrojivih tipova podataka, kao što je slučaj

type Dan=(Pon,Uto,Sre,Cet,Pet,Sub,Ned)

promenljive tipa Dan se predstavljaju od strane kompilatora pomoću celobrojne vrednosti od 0 do 6.

3.2.2. Booleans

To su najjednostavniji nabrojivi tipovi podataka, koji imaju samo dve vrednosti. Ovi tipovi se definišu u mnogim jezicima kao

type Boolean=(False,True)

3.2.3. Znaci

Znaci su takođe nabrojivi tipovi podataka. Za razmenu informacije između računara potreban je skup podataka. Skup znakova se komponuje shodno specifičnim pravilima. Skup znakova mora da sadrži slova, znake interpukcije, komercijalne simbole i aritmetičke operatore. Druga klasa simbola koji pripadaju skupu znakova su upravljački znaci (*control characters*).

U Tabeli 3.1 prikazan je skup znakova koji pripadaju ASCII kodu (*American Standard Code for Information Interchange*), koji je formalno poznat kao ISO kod.

Tab. 3.1. ASCII kod.

	Viši deo							
Niži deo	**000**	**001**	**010**	**011**	**100**	**101**	**110**	**111**
0000	NUL	DLE	SP	0	Ž	P	ž	p
0001	SDH	DC1	!	1	A	Q	a	q
0010	STX	DC2	"	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	Š	k	š
1100	FF	FS	,	<	L	Đ	l	đ
1101	CR	GS	-	=	M	Ć	m	ć
1110	SO	RS	.	>	N	Č	n	č
1111	SI	US	/	?	O	_	o	DEL

Primer: ASCII kod za znak '>' je 0b0111110.

Skraćenice za upravljačke kodove			
ACK	acknowledge	**ETX**	end of text
BEL	bell	**FF**	form feed
BS	backspace	**FS**	file separator
CAN	cancel	**GS**	group separator
CR	carriage return	**HT**	horizontal tabulation
DC1	device control 1	**LF**	line feed
DC2	device control 2	**NAK**	negative acknowledge
DC3	device control 2	**NUL**	null
DC4	device control 4 (stop)	**RS**	record separator
DEL	delete	**SI**	shift in
DLE	data link escape	**SO**	shift out
EM	end of medium	**SOH**	start of heading
ENQ	enquiry	**STX**	start of text
EOT	end of transmission	**SUB**	substitute
ESC	escape	**SYN**	synchronous idle
ETB	end of transmission block	**US**	unit separator
		VT	vertical tabulation

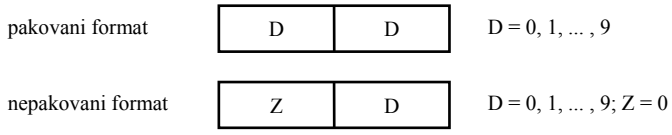
3.2.4. Brojevi u fiksnom zarezu

Ovi brojevi se često koriste u administrativnim aplikacijama, jer ovaj tip podataka veoma blisko odgovara konceptualnom načinu predstavljanja podataka od strane krajnjeg korisnika. Obično se za prezentaciju koriste decimalni brojevi. Za implementaciju decimalnih brojeva (koje čine decimalne cifre) sledeći su aspekti važni:

Brojevi se kodiraju u BCD kodu,

- Kod notacije znak moduo moguće je koristiti nekoliko notacija,
- Broj decimalnih cifara,
- Broj cifara po svakom bajtu - postoje dve verzije, a to su pakovani BCD broj i nepakovani BCD broj.

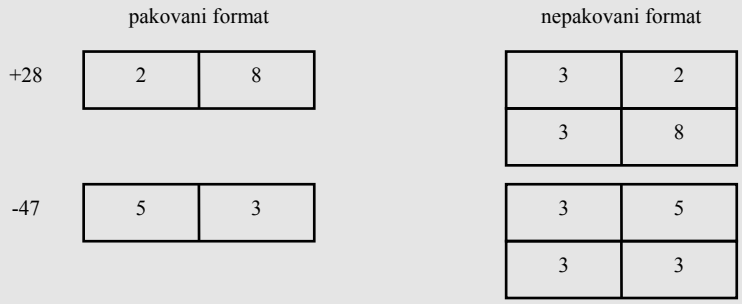
Kod pakovanog decimalnog formata svaki bajt se deli na dva četvorobitna polja od kojih svako sadrži BCD cifru. Kod nepakovanog decimalnog formata, svaki bajt se deli na dva četvorobitna polja; polje manje težine sadrži cifru od 0 do 9 a polje veće težine je nula.



Kombinacija Z-D često predstavlja i znak koji se može štampati. Vrednost "3" kod ASCII i "F" kod EBCDIC koda se često koriste za polje Z.

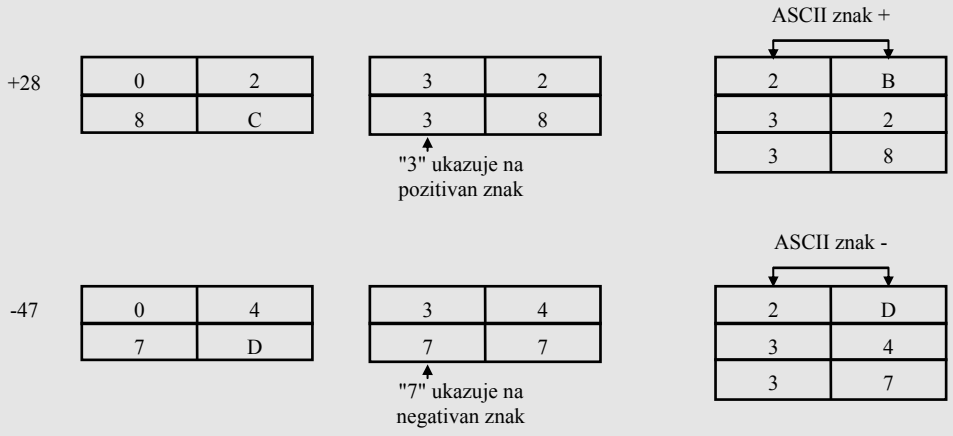
Primer 3.1:

Kod mikroprocesora MC68020 koriste se pakovani i nepakovani decimalni format. Negativni brojevi se predstavljaju u desetičnom komplementu.



Sa druge strane, VAX-11 podržava tri različita formata, od kojih su dva nepakovana. Pakovani decimalni format koristi prezentaciju znak-moduo gde se znak kodira u saglasnosti sa tabelom i postavlja se na LS tetradu (niblu).

Cifra/znak	Binarni kod	Heksa kod
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9
+	1010, 1100, 1110, 1111	A, C, E, F
-	101,1101	B, D



3.2.5. Brojevi u pokretnom zarezu

Kod najvećeg broja HLL-ova postoji mogućnost deklarisanja promenljive tipa real. U računarima se brojne vrednosti ovog tipa predstavljaju u notaciji pokretnog zareza.

$$V=(-1)^s \times F \times R^E,$$

gde je:

S-znak; F-mantisa; E-eksponent; R-brojna osnova.

U računarskim sistemima se koristi nekoliko formata realnih brojeva.

Normalizovana notacija

Nedostatak FP (*floating point*) notacije je što se brojevi mogu predstaviti na nekoliko načina, tj. ne postoji jedinstvena predstava za jedan broj.

Primer 3.2:

37.145 se može predstaviti kao 0.0037145×10^3 ili kao 3714.5×10^{-2}.

Ovi brojevi se zovu *nenormalizovani*. *Normalizovani* su oni brojevi kod kojih mantisa zadovoljava uslov:

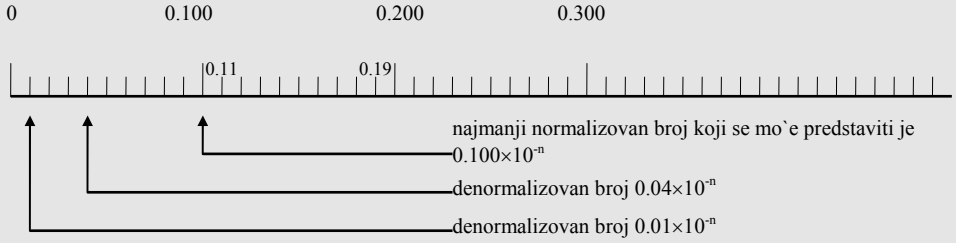
$$1>F\geq 1/R$$

Primer 3.3:

U normalizovanoj formi broj 37.145, za R=10, se predstavlja kao 0.37145E+2

Naglasimo da normalizovana notacija ne obezbeđuje predstavljanje određenih brojeva blizu nule. Na slici 3.3 prikazani su normalizovani brojevi 0.3E-n, 0.2E-n, 0.19E-n, 0.11E-n i 0.1E-n gde zadnji predstavlja najmanji normalizovan broj. Brojevi manji od normalizovanog broja zovu se *denormalizovani* brojevi i ne zadovoljavaju uslov $1>F\geq 1/R$.

Denormalizovani brojevi kao što su 0.04E-n, 0.01E-n i 0.001E-n se mogu tada predstaviti.



Sl. 3.3. Normalizovani i denormalizovani brojevi.

Opseg

Opseg zavisi od broja bitova koji su dostupni eksponentu i brojnoj osnovi. Opseg je ograničen sa jedne strane najmanjim brojem, a sa druge strane najvećim brojem koji se može predstaviti. Shodno tome, u jednom slučaju kažemo da kada radimo sa malim brojevima imamo podbačaj (underflow), a u drugom, kada radimo sa velikim može se javiti premašaj (overflow). Obično se premašaj signalizira, a podbačaj konvertuje u nulu.

Preciznost

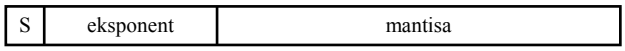
Postoji i drugi skup brojeva koji se ne može predstaviti, jer se ti brojevi nalaze između brojeva koji se mogu predstaviti. Na primer, ako mantisa sadrži tri cifre, tada se 0.3114×10^{-2} ne može predstaviti i aproksimira se sa 0.311×10^{-2}. Preciznost zavisi od broja dostupnih bitova mantise.

Brojna osnova

Najveći broj računara koristi brojnu osnovu 2. Kod IBM/370 koristi se brojna osnova 16 iz razloga efikasnosti - smanjuje se broj operacija pomeranja, tj. normalizacija se vrši samo ako su sva četiri MS bita frakcije jednaki nuli.

Struktura FP formata

Format FP broja prikazan na slici 3.4 usvojen je od najvećeg broja arhitektura



Sl. 3.4. Format FP broja.

gde je S - znak broja (S=0 pozitivan, S=1 negativan). Eksponent može biti zadat u modifikovanoj i nemodifikovanoj notaciji. Kod nemodifikovane notacije (*unbiased notation*) vrednost eksponenta je u opsegu od -128...0...+127 (za 8-bitni eksponent). Kod modifikovane notacije (*biased notation* ili *exess notation*) vrednost eksponenta je u opsegu 0...+128...+255.

Predstavljanje broja nula

Predstavljanje nule u FP notaciji može biti problem. Može se reći da se bilo koja prezentacija kod koje mantisa na svim cifarskim pozicijama ima nulu interpretira kao nula. Poželjno je takođe da u tom slučaju i polje eksponent bude nula.

Nevidljivi bit

Kada je osnova 2, a brojevi su uvek normalizovani, MS bit mantise će uvek biti jedan (sa izuzetkom kada je vrednost nula). Ovo znači da prva jedinica ne nosi bilo kakvu informaciju i zbog toga ne mora da se pamti, što omogućava da se mantisi može dodeliti još jedan dodatni bit. Kada se vrši izračunavanje o ovom bitu treba voditi računa i imati u vidu da on postoji, a kada se rezultat želi sačuvati (memorisati) njega ne treba upisivati. Ovaj bit se zove *nevidljivi bit* (*hidden bit*). Tehnika rada sa nevidljivim bitom se često koristi kod velikog broja arhitektura, kod kojih specijalni hardver (FP procesor) mora da uzme u obzir nevidljivi bit kada interno obavlja konverziju podataka.

Standardi za FP

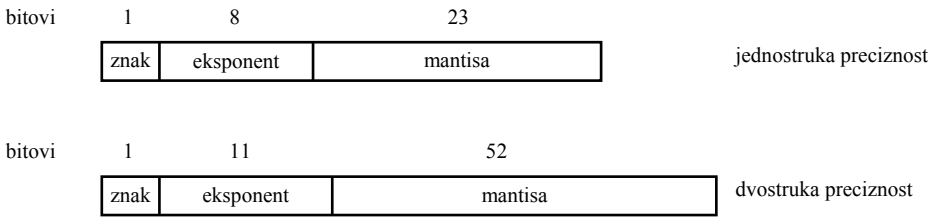
Postoji veći broj standarda za FP od kojih su najpoznatiji IEEE Standard 754, IBM-ov standard korišćen kod mašine IBM/370 i DEC-ov standar korišćen kod mašine PDP-11.

IEEE Standard 754 je važeći kako za hardversku, tako i za softversku implementaciju FP operacija. Standardom se razmatraju problemi koji su u vezi sa zaokruživanjem (rounding), premašajem, podbačajem i drugim rezultatima koji se javljaju kao izuzetak u toku izračunavanja.

Kod IEEE 754 postoje četiri osnovna formata:

- jednostruka preciznost E=8 bitova, F=23 bita,
- dvostruka preciznost E=11 bitova, F=52 bita,
- jednostruka preciznost - prošireni format E≥11 bitova, F≥31 bita,
- dvostruka preciznost - prošireni format E≥15 bitova, F≥63 bita.

Tačne vrednosti za polja E i F zavise od implementacije. IEEE FP formati su prikazani na slici 3.5.



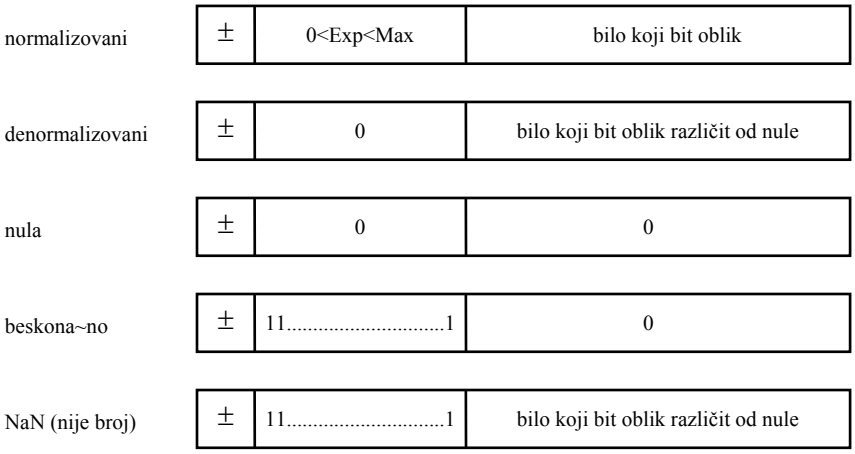
Sl. 3.5. IEEE FP formati za jednostruku i dvostruku preciznost.

Numeričke karakteristike IEEE FP standarda date su Tabeli 3.2.

Tab. 3.2. Numeričke karakteristike IEEE FP standarda.

stavka	jednostruka preciznost	dvostruka preciznost
bit znaka	1	1
bitovi eksponenta	8	11
bitovi mantise	23	52
ukupno bitova	32	64
sistem eksponenta	modifikovan za 127	modifikovan za 1023
opseg eksponenta	-126 do + 127	-1022 do + 1023
najmanji normalizovani	2^{-126}	2^{-1022}
najveći normalizovani	aproksimativno 2^{+128}	aproksimativno 2^{+1024}
decimalni opseg	aproksimativno od 10^{-38} do 10^{+38}	aproksimativno od 10^{-308} do 10^{+308}
najmanji denormalizovani	aproksimativno 10^{-45}	aproksimativno 10^{-324}

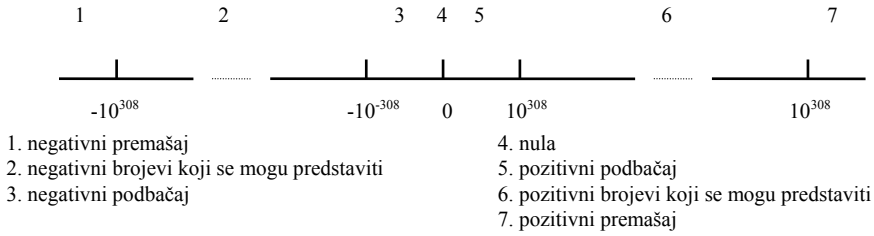
Pregled numeričkih tipova podataka kod IEEE standarda prikazan je na slici 3.6.



Sl. 3.6. Numerički tipovi podataka kod IEEE standarda.

NaN se javlja kada se beskonačnost deli sa beskonačnošću. Da bi se obradio ovaj slučaj usvaja se poseban format NaN (*Not a Number*).

Brojna linija na kojoj se vrši prezentacija ralnog broja u duploj preciznosti se može predstaviti u sedam oblasti.



1. negativni premašaj
2. negativni brojevi koji se mogu predstaviti
3. negativni podbačaj
4. nula
5. pozitivni podbačaj
6. pozitivni brojevi koji se mogu predstaviti
7. pozitivni premašaj

Sl. 3.7. Oblasti brojne ose koje se mogu i koje se ne mogu predstaviti.

3.3. Strukturni tipovi podataka

Strukturni tip podataka čini skup elemenata podataka koji su u međusobnoj vezi. Elementi podataka mogu biti skalarne veličine ili strukturni podaci. Struktura podataka je fiksirana u toku kompilacije i ne može se menjati. Na osnovu ovoga jasno je da strukturni tipovi podataka mogu biti veoma kompleksni, pa je to i razlog što ih mali broj računara podržava. Od strukturnih tipova podataka analiziraćemo:

- skupove,
- polja,
- nizove znakova,
- zapise.

3.3.1. Skupovi

Skup, kao što se koristi kod Pascala, je mnoštvo vrednosti. Ove vrednosti čine sve podskupove nabrojivih elemenata (uključujući prazan skup) gde elementi imaju samo skalarne tipove podataka ili pripadaju podopsegu. Na primer, skup {'A',...,'Z','0',...,'9'} sadrži {'A'}, {'C', 'D', '3', '7'}. Za skup tipova podataka definišu se specifične operacije:

*	postavi presek
+	postavi uniju
-	postavi diferencu
in	postavi član

Bit vektori često se koriste za implementaciju skupova. Kompilator je odgovoran za prevođenje svakog člana skupa na bit u vektoru. Logičke operacije se zatim izvode nad bit vektorima.

Primer:

Definišimo sledeće tipove promenljivih:

```
type Dan=(Pon,Uto,Sre,Cet,Pet,Sub,Ned);
var D : set of Dan;
```

Promenljiva D se može implementirati pomoću 7-bitnog vektora, kao što je prikazano na slici 3.8.

7	6	5	4	3	2	7	0
-	Ned	Sub	Pet	Cet	Sre	Uto	Pon

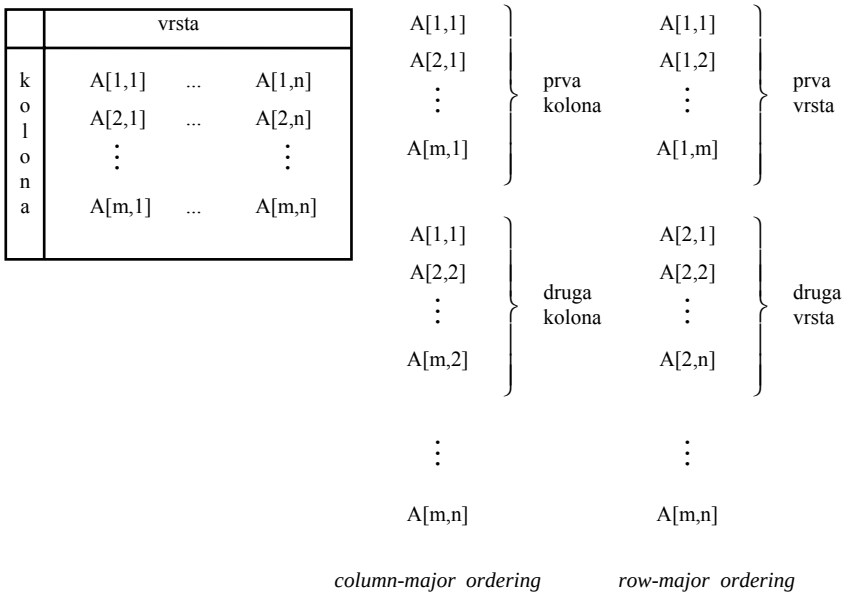
Sl. 3.8. Implemetacija promenljive tipa set.

Podskup Radni_dani se definiše kao podskup {Pon,...,Pet}. Test "Sub in Radni_dani" se može implementirati pomoću logičke funkcije AND nad dvobitnim vektorima, koji predstavljaju podskup Sub i Radni_dani. Sub se predstavlja pomoću bit vektora koji ima 1 na bit poziciji 5 a Radni_dani imaju 1 na bit pozicijama 0-4. Rezultat AND operacije je 0 što znači da Sub, na sreću, nije radni dan.

3.3.2. Vektori i polja

Normalno, vektor je jednodimenzionalna, a polje multidimenzionalna kolekcija elemenata podataka istog tipa, pri čemu se svaki element može selektivno izdvojiti koristeći se jednim ili većim brojem indeksnih vrednosti. Polje se smešta u memoriju tako da se elementi polja smeštaju u kontinualnememorijske oblasti, na jedan od sledeća dva načina (slika 3.9):

- uređenje po koloni (*column-major ordering*) - ovaj metod se koristi kod FORTRAN-a.
- uređenje po vrsti (*row-major ordering*) - ovaj metod smeštanja podataka u memoriju koriste skoro svi jezici osim FORTRAN-a.



Sl. 3.9. Smeštanje dvodimenzionalnog polja u memoriju po kolonama i po vrstama.

3.3.3. Znakovni nizovi

Znakovni niz je sekvencijalni niz znakova. Kako se tekst izvornog programa može smatrati nizom znakova, operacije sa nizovima su veoma važne za kompilatore i asemblere, kao i za tekst procesore. Često se znakovni nizovi mogu razmatrati kao polja nizova što čini da svaki znak bude direktno adresibilan.

3.3.4. Zapisi

Zapis je skup elemenata podataka, pri čemu svaki može da pripada različitom tipu podataka, što je u suprotnosti sa definicijom polja. Elementi podataka se zovu *polja*, adresiraju se posebno odgovarajućom labelom, tzv. *identifikatorom polja*. Element podataka se selektuje imenom zapisa, zajedno sa identifikatorom polja. Polje se zatim može deliti na potpolja pri čemu svako ima svoje ime. Potpolja se mogu dalje deliti itd.

Kao struktura, zapis se deklariše u toku kompilacije. Zapis se ne može menjati u toku izvršenja programa. Shodno tome, pomeraj (*offset*) za svako polje, relativno u odnosu na početak zapisa, je fiksan i izračunava se u toku kompilacije.

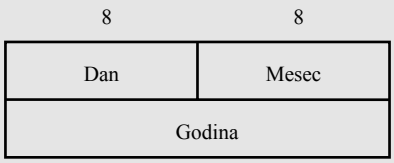
Da bi se ostvarila veća fleksibilnost, moguće je uvesti promenljivi deo u definiciju. Ovaj promenljivi deo dinamički definiše strukturu zapisa, tj. promenljivi deo sadrži nekoliko mogućih (različitih) definicija, od kojih se samo jedna bira za izvršenje. Pri ovome se uključuje *tag* polje koje se koristi za biranje jedne od definicija zapisa.

Primer 3.5:

U sledećem programu, promenljiva tipa Datum ima tri polja: Dan, Mesec i Godina. Polje u zapisu se može specificirati pomoću zapisa ime, nakon kojeg sledi period i ime polja.

```
type    Datum = record
                  Dan:1..31;
                  Mesec:1..12;
                  Godina:integer;
                end;
var     Rodjendan:Datum;
begin
                Rodjendan.Godina:=1964;
                Rodjendan.Mesec:=12;
                Rodjendan.Dan:=8;
end
```

Zapis će biti alociran kao na slici



gde svako polje ima fiksnu poziciju u odnosu na početak zapisa.

3.4. Tipovi podataka za pristup

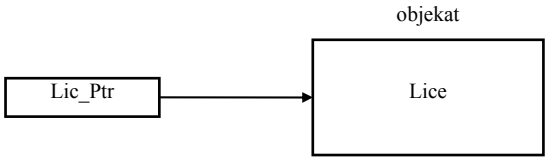
Ovi tipovi podataka otvaraju mogućnost kreiranja objekata podataka i struktura dinamički, tj. u toku izvršenja programa. Identifikatori ovog tipa se deklarišu na početku programskog modula. Kod Pascala oni se zovu pokazivači (*pointers*); tip podataka pokazivača se smatra da je tip podataka objekta na koji pokazivač ukazuje. Samo memorija koja je potrebna za pokazivač se alocira, a aktivira se blok koji sadrži pokazivač. Na primer, ako je tip podatka Lice već deklarisan, tada definicija

```
type    Lic_Ptr_type : ^Lice;
```

označava da je Lic_Ptr_type tip pristupa koji ukazuje na objekat tipa Lice. Ako je Lice tipa zapis, tada deklaracija

```
var     Lic_Ptr : Lic_Ptr_type;
```

označava pokazivač (nazvan Lic_ptr) koji ukazuje na zapis sa istom strukturom kao što je definisana u Lice .



Na žalost, ova deklaracija kreira samo pokazivač na objekt (Lice_Ptr), a ne i objekt podataka na koji se ukazuje (zapis tipa Lice).

Kreiranje dinamičkih objekata se izvodi u toku vremena izvršenja iskaza. Kod Pascala se za ovo koristi standardna procedura **new**. Oslobađanje alociranog memorijskog prostora kod Pascala se obavlja pomoću procedura **free** ili **dispose** (ime zavisi od verzije Pascala).

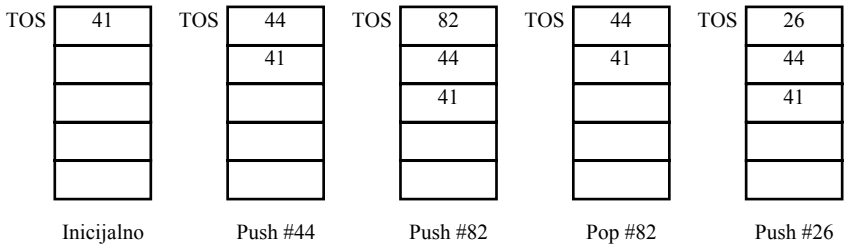
Mašinska predstava pokazivača je normalno adresa. Kod najvećeg broja arhitektura, aritmetika nad adresama se obavlja preko celobrojnih operacija. U tom slučaju, celobrojni tip podataka ima istu dužinu kao i adresa. Šta više, da bi pristupili objektima preko pokazivača, važno je da arhitektura podržava indirektno adresiranje.

3.5. Sistemski strukturni tipovi podataka

Koriste se od strane operativnog sistema za upravljanje programom unutar računara. Magacin (*stack*) se često koristi za smeštaj (čuvanje) stanja procesora (zadataka) kod kontekst komutacije ili za aktiviranje zapisa u toku izvršenja programa. Redovi čekanja (*queues*) se često koriste da formiraju bafere, na primer, proces koji čeka obično se planira da bude izvršen od strane procesora, ili čeka na U/I.

3.5.1. Magacini

Magacin tipa guranje naniže (push-down stack) se može posmatrati kao struktura podataka kod koje vrednosti ulaze ili se izbavljuju samo sa jednog kraja. Ćelija koja sadrži zadnje unetu vrednost poznata je kao vrh magacina (*top of stack* - TOS). S obzirom da se stavka koja je uneta zadnja, prva izbavlja, magacin se zove LIFO bafer (*Last-In-First-Out Buffer*). Operacija upisa u magacin se zove **push** a operacija čitanja **pop**. Na slici 3.10 prikazan je magacin kod izvođenja osnovnih **push** i **pop** operacija.



Sl. 3.10. Primer magacina i izvođenja operacija **push** i **pop**.

Tehnika opisana na slici 3.10 zahteva da se svi elementi u magacinu premeštaju kada se izvršaju **push** i **pop** operacije. Magacin se obično implementira preko pokazivača magacina, koji uvek ukazuje na vrh magacina (TOS). Magacin je tada kontinualni memorijski blok, pa kada se obavlja operacija sa magacinom, pokazivač se inkrementira ili dekrementira, dok elementi ostaju u istim memorijskim lokacijama.

U toku izvršenja programa mora se voditi računa da magacin ne premaši alociranu (dodeljenu) memorijsku oblast, jer će to dovesti do gubitaka nekih elemenata ili destrukcije susednih struktura podataka ili programa. Ovaj efekat zove se premašaj. Druga opasna situacija je podbačaj, a javlja se kada se pokušava izvršiti operacija **pop** a magacin je prazan. Ako se ovaj slučaj ne detektuje mogu se javiti nepredvidljivi rezultati.

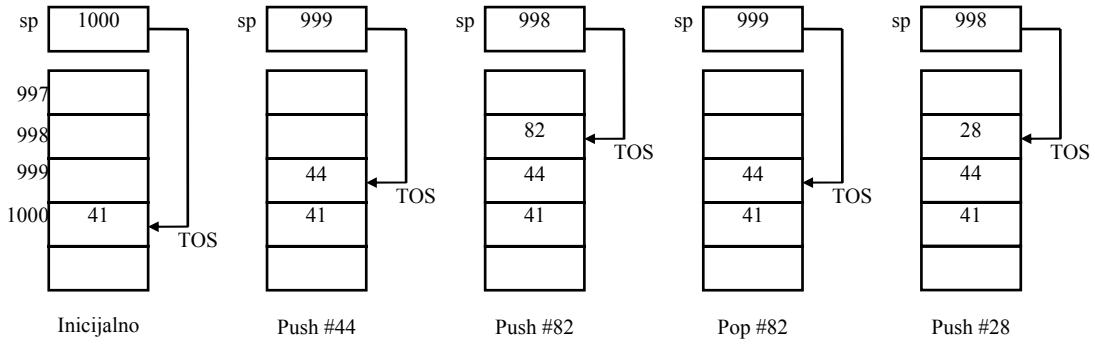
Za magacin kažemo da raste naniže. Operacija smeštanja elemenata u magacin se obavlja u sledećem redosledu:

1. smanjuje se pokazivač magacina, tj. ukazuje na praznu lokaciju,
2. vrši se smeštanje podataka u magacin.

Operacija čitanja podataka iz magacina se obavlja sledećim redosledom:

1. čita se podatak iz magacina,
2. povećava se sadržaj pokazivača magacina a njegova nova vrednost ukazuje na novu lokaciju.

Na slici 3.11 prikazan je magacin i pokazivač magacina SP (stack pointer) u toku izvršenja nekoliko PUSH i POP instrukcija.

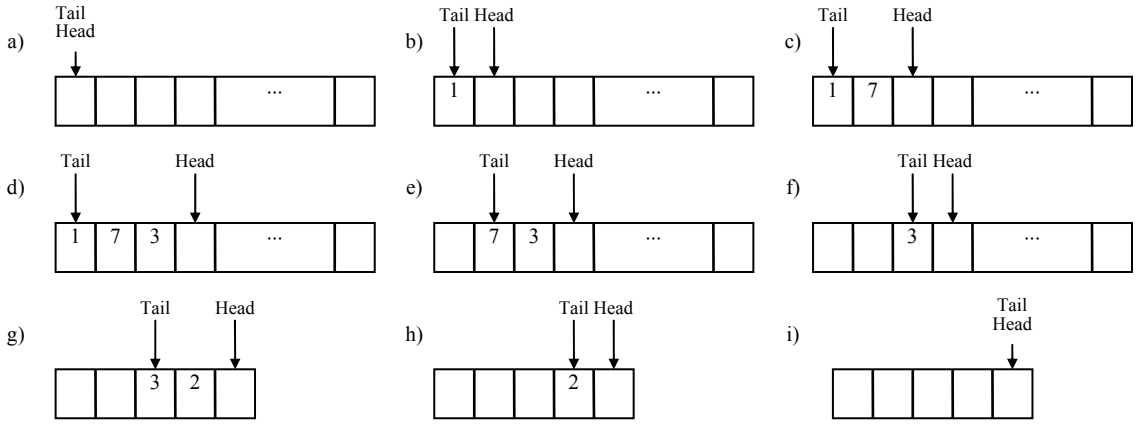


Sl. 3.11. Primer magacina i pokazivača magacina kod izvođenja operacija **push** i **pop**.

3.5.2. Redovi čekanja

Drugi važan tip podataka je red čekanja. Red čekanja je FIFO (First-In-First-Out) bafer. Drugim rečima, podatak koji je smešten prvi izbavlja se kao prvi. Redove čekanja koristi operativni sistem za čuvanje opisa procesa koji treba da se izvrši. Procesi se izbavljaju iz redova čekanja kada su spremni za izršenje.

Nasuprot magacinu, kod redova čekanja potrebna su dva pokazivača: prvi ukazuje na početak (*head*), a drugi na kraj (*tail*). Pokazivači ukazuju na lokacije gde se podaci unose i izbavljaju respektivno. U suštini, "*head*" pokazivač ukazuje na prvu slobodnu lokaciju nakon ulaza koji je unet kao zadnji. U slučaju da oba "*head*" i "*tail*" pokazivača ukazuju na istu lokaciju, za red čekanja kažemo da je prazan. Korišćenje ovih pokazivača, kao i korišćenje operacija tipa **enqueue** i **dequeue** kojima se manipuliše redovima čekanja, prikazano je na slici 3.12.



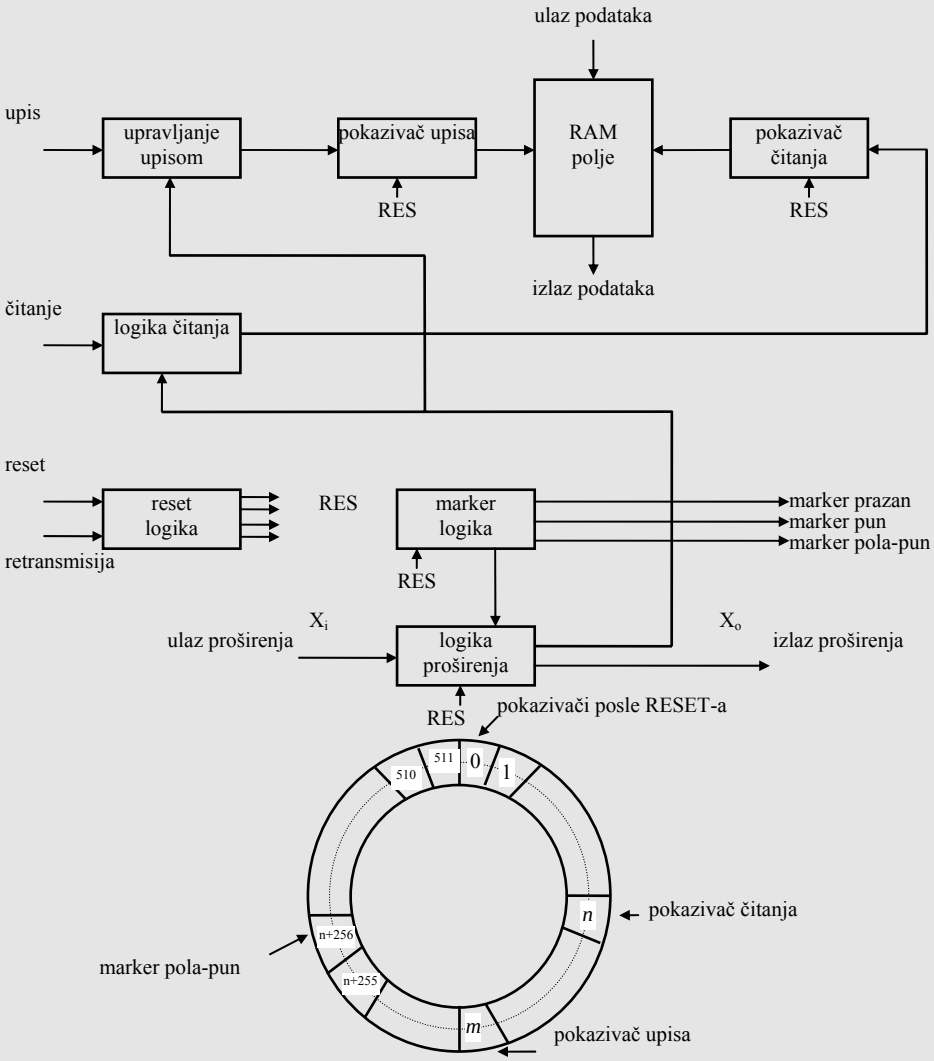
Sl. 3.12. Primer reda čekanja.

Operacije prikazane na slici 3.12. su sledeće:

a) inicijalno prazan red čekanja, b) enqueue #1, c) enqueue #7, d) enqueue #3, e) dequeue #1,
f) dequeue #7, g) enqueue #2, h) dequeue #3, i) dequeue #2 (red čekanja je opet prazan).

Primer 3.6:

Blok šema implementacije FIFO memorije prikazana je na slici 3.13.



Sl. 3.13. FIFO memorija - blok šema.

Princip rada FIFO memorije sa slike 3.13 je sledeći. Centralni deo FIFO memorije je RAM polje. Dva pokazivača "*Read-pointer*" i "*Write-pointer*" pokazuju na narednu lokaciju iz koje se čita ili upisuje. Polje ima dva posebna ulaza i dva brojača koji se nezavisno taktuju, što obezbeđuje da se simultano vrši upis i čitanje sa različitim brzinama. Kada se FIFO resetuje oba brojača se postavljaju na 0, a zatim inkrementiraju sa svakim impulsom čitanja ili upisa. Brojači se ne zaustavljaju na kraju fizičke memorije. Kada prime maksimalnu vrednost ponovo počinju od 0 i inkrementiraju se. Svaki brojač zaustavlja inkrementiranje kada dostigne vrednost drugog brojača. Kada "*Read pointer*" dostigne vrednost "*Write pointer*", FIFO je prazan, a kada "*Write-pointer*" dostigne vrednost "Read-pointer", FIFO je pun. Apsolutna vrednost brojača je irelevantna.

Marker logika nadgleda razliku između oba pokazivača i u saglasnosti sa tim postavlja markere. Kada se pokazivači upare (izjednače), logika aktivira marker prazan ili marker pun. Kada razlika između pokazivača premaši dubinu pola RAM polja, logika postavlja marker pola-pun. Logika proširenja obezbeđuje mehanizam lančanog vezivanja (*daisy-chaining*) FIFO memorija kada se želi postići veća dubina polja. U zavisnosti od stanja ulaza X_i, FIFO memorija deluje kao jedinstveni sklop ili kao deo lanca. Druga linija FL (*First Load* - nije prikazana na slici 3.13) prikazuje koja će FIFO memorija u lancu nakon reseta prva prihvatiti podatak. Kada je FIFO memorija u lancu puna, ona u toku ciklusa UPIS aktivira izlaz X_o, čime ukazuje narednom sklopu u lancu da može da memoriše podatak.

Kada je povezivanje izvedeno kružno, X_o jednog sklopa povezuje se sa X_i narednog, a FIFO se može predstaviti kao jedinstvena memorija velikog kapaciteta.

Komanda RETRANSMIT postavlja pokazivač čitanja na početku adresnog prostora, pa se podaci mogu ponovo čitati sa tog mesta. Problem se javlja kada se vrši čitanje i upis sekvence blokova podataka. Čitanje podataka iz FIFO-a nema efekat na pokazivač upisa. Ako treba čitati neki od podataka u FIFO, pokazivač upisa produžava sa inkrementiranjem iznad kraja fizičke memorije (skače sa vrednosti maksimalnog modula brojanja na nulu) na početak. FIFO memorija je jedino puna ako su oba brojača jednaka; njihove apsolutne vrednosti su irelavantne.

Ako ukupan broj upisanih reči u FIFO premašuje njegov kapacitet i pored toga što se pomoću izvođenja operacija čitanja obezbeđuje da se FIFO ne popuni, korišćenjem RETRANSMIT komande vraćaju se ponovo u FIFO podaci koji se izbacuju kada pokazivač prelazi sa maksimalnog modula na nulu (*wrap-around* - WA). WA se može sprečiti resetovanjem FIFO memorije, čime se oba pokazivača postavljaju na nulu.