Sveučilište u Splitu
Fakultet elektrotehnike, strojarstva i brodogradnje

Bojan Vujatović

PROVJERA ZAPISA REALNIH BROJEVA U 32-BITNI REGISTAR POMOĆU PROGRAMA

seminarski rad

Kolegij: Uvod u računala i programiranje

Mentorica: dr. sc. Mirjana Bonković

Split, prosinac 2011.

SADRŽAJ

1. UVOD

Za uspješno izvršenje gotovo svih programa koji se izvode na računalima potrebno je koristiti realne brojeve, odnosno realne varijable. Od velike važnosti je zapisivanje brojeva uz što manje odstupanje od stvarne vrijednosti broja, odnosno uz što manju grešku zapisa. Samo jedan od primjera koji potvrđuju ovu činjenicu je računanje položaja, smjera kretanja i drugih podataka prilikom plovidbe ili upravljanja zrakoplovom, gdje su i male greške nedopustive.

No, računalno je ograničeno, te može zapisati samo određeni broj realnih (i cijelih) brojeva. Cijela beskonačnost ostatka brojeva se mora „stisnuti“ u registar memorije. Broj $12.4_{(10)}$ je jedan od tih. Budući da nema konačan zapis u binarnom brojevnom sustavu, neće moći biti točno zapisan u računalu nego samo približno, odnosno bit će zaokružen na najbliži broj koji koji se može točno zapisati.

Cilj ovoga rada je pronaći način (algoritam) na koji računalo u memoriji zapisuje realne brojeve jednostruke preciznosti (32 bita, u programskom jeziku C tip `float`). Točnije, pronaći algoritam zaokruživanja, jer je poznato da je upis u registre reguliran IEEE standardom. Uz algoritam slijedi i pripadajući program napisan u programskom jeziku C (u programskom okruženju MS Visual Studio 2010).

2. ZAPIS REALNIH BROJEVA U REGISTRE

2.1. Normalizirani oblik broja

Svaki broj, u nekoj bazi b, može se prikazati u obliku:

$$\pm a_0.a_1a_2a_3 \ldots \cdot b^n,$$

gdje su znamenke $a_0, a_1, a_2, a_3, \ldots \epsilon \{0, 1, \ldots, b-1\}$, $a_0 \neq 0$, a $n \epsilon Z$. Ovakav oblik se naziva normalizirani oblik broja. Broj n se naziva potencija normaliziranog zapisa broja, a skup znamenki $a_1a_2a_3$ ... mantisa broja.

Računalo radi na binarnom brojevnom sustavu, pa se broj pretvara u normalizirani oblik u binarnom sustavu. No tada znamenka a_0 može poprimiti samo vrijednost 1, pa vrijedi ovakav zapis:

$$\pm 1.a_1a_2a_3 \ldots \cdot 2^n,$$

a $a_1, a_2, a_3 \ldots \epsilon \{0, 1\}$.

Važno je napomenuti da konačni decimalni brojevi u dekadskom sustavu ne moraju imati konačan zapis u drugim sustavima, kao ni u binarnom. Drugim riječima mantisa broja u dekadskom sustavu može biti konačna, a mantisa istog broja u binarnom sustavu ne. Primjer:

$$12.4_{(10)} = 1100.0110011001100110011001100110011001100110011001100 \ldots_{(2)}$$

$$= 1.1000110011001100110011001100110011001100110011001100 \ldots_{(2)} \cdot 2^3$$

Upravo ova činjenica stvara velike probleme prilikom zapisa u registre.

2.2. IEEE standard

Realni brojevi se u registre po zapisuju po IEEE standardu. Taj standard propisuje IEEE (Institute of Electrical and Electronics Engineers), a trenutno najnovija verzija je IEEE 754-2008. Prema njoj, brojevi u 32-bitnom registru se zapisuju ovako:

Slika 1. – Zapis broja po IEEE standardu

P označava jedan bistabil (bit) za predznak, sljedećih osam bitova (EXP) su bitovi za predstavljanje ekponenta, a ostala 23 bita – bitovi za mantisu broja.

U bit za predznak se upisuje 0 ako je broj pozitivan, a 1 ako je broj negativan.

Eksponent se računa kao:

$$EXP_{(10)} = POTENCIJA_{(10)} + POMAK_{(10)}.$$

Pomak za registar od 32 bita iznosi $127_{(10)}$. Na kraju eksponent se kodira tako da se dekadski ekvivalent pretovi u binarni te zapiše tako da zauzme ukupno 8 mjesta (manjak znamenki se nadoknadi tako da se sa desne strane niz nadopuni nulama).

U preostala 23 bita za mantisu upisuje se mantisa broja. U slučaju da broj ima konačan zapis u binarnom sustavu te njegova mantisa je duljine manje od 23 znamenke, upisat će se broj jednak početnom, odnosno čitava mantisa, a ostatak će se nadopuniti nulama s lijeve strane. U suprotnome zapisat će se prve 23 znamenke mantise i javit će se odstupanje koje će računalo pokušati minimizirati.

Postoje i odstupanja od ovih pravila kodiranja koja su dogovrena, a to su zapisi jako malih brojeva za čiji zapis koristimo denormalizirani oblik, te zapisi nule i NaN (Not a Number) vrijednosti. Ovaj rad ne pokriva takve slučajeve budući da je za predstavljanje velike većine brojeva dovoljan i standardan zapis.

2.3. Postupak provjere broja

Računalo tada provjerava da li je iskoristilo registar za prikaz početnog broja na najbolji mogući način. Pokažimo to na primjeru broja $12.4_{(10)}$:

$$12.4_{(10)} = 1.10001100110011001100110011001100110011001\ldots_{(2)} \cdot 2^3$$

P – 0 (pozitivan broj)

EXP – $3_{(10)} + 127_{(10)} = 130_{(10)} = 10000010_{(2)}$

MANTISA – $10001100110011001100110011001100110011001\ldots$

Dakle, zapis u registru je:

0	1	0	0	0	0	0	1	0	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0

No, pogledajmo koji je zapravo broj zapisan:

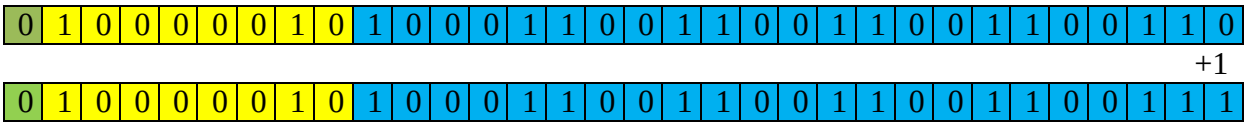
$$1.10001100110011001100110_{(2)} \cdot 2^3 \approx 12.399999618530273_{(10)}$$

Apsolutna greška koja nastaje zbog ovog zapisa jednaka je:

$$\Delta_1 \approx |12.399999618530273 - 12.4| \approx 0.000000381469727$$

Broj zapisan na ovaj način je sigurno manji od početnog broja ili u posebnom slučaju jednak (ako se broj uspio zapisati), budući da odbacujemo znamenke s kraja mantise.

No valja provjeriti postoji li broj koji se može upisati, a da je apsolutna greška takvog broja manja od Δ_1. Ako postoji, to je sigurno najmanji broj koji je veći od prvotno zapisanog, odnosno njegov sljedbenik. Možemo ga nazvati sljedbenikom jer je skup brojeva koji se mogu zapisati u registar konačan i diskretan. Njega ćemo dobiti tako da na postojeću mantisu zbrojimo jedinicu:



Sada je zapisan broj:

$$1.10001100110011001100111_{(2)} \cdot 2^3 \approx 12.400000572204590_{(10)},$$

$$\Delta_2 \approx |12.400000572204590 - 12.4| \approx 0.000000572204589$$

Vidimo da je $\Delta_1 < \Delta_2$ pa će u memoriji ostati zapisan prvi broj. To možemo vidjeti i na slici 2., ako ispišemo broj $12.4_{(10)}$ u formatu `float`:

```
printf(„\n\n Stvarni zapis broja: %.15f\n“, float_broj);
```

No to ne mora biti slučaj uvijek. Vidimo na drugom primjeru, pri upisu broja $3.14_{(10)}$ da će vrijediti $\Delta_1 > \Delta_2$, pa će u memoriji biti zapisan broj $3.140000104904175_{(10)}$.

3. IMPLEMENTACIJA U PROGRAMSKOM JEZIKU C

U ovoj implementaciji korištene su datoteke `stdio.h`, `stdlib.h`, `string.h` te uglavnom osnovne funkcije: funkcije za standardni upis i ispis te rad sa stringovima. Rad s brojevima zapisanim u stringu je potreban jer bi računanje s `float` ili `double` varijablama generiralo grešku na grešku u računu, što želimo izbjeći.

U glavnoj, `main()` funkciji vrti se petlja glavnog izbornika koja se prekida kada korisnik unese broj za izlaz iz programa. U glavnom izborniku je moguće odabrati opciju za predviđanje zapisa broja izračunatog u funkciji `void IEEE(char *broj)` funkciji i opciju za ispis nekog broja zapisanog u `float` varijabli.

Druga opcija je riješena u malo linija koda. Naprosto je potrebno upisati broj u varijablu te je ispisati sa svim svojim znamenakama:

```
printf(„\n\n Stvarni zapis broja: %.15f\n“, float_broj);
```

Za izvedbu prve opcije bilo je potrebno utrošiti mnogo više linija koda. Kao što je već spomenuto, cijeli postupak se vrši u funkciji `void IEEE(char *broj)` koja po potrebi poziva pomoćne funkcije za pomoćne radnje.

Prije pozivanja same funkcije upisuje se željeni broj u string te se osigurava da je pravilno upisan prolaskom kroz funkciju `int valjanost_unesenog_broja(char *broj)`. U slučaju da broj nije valjan, može se pokužati ponovo.

Prvo je potrebno provjeriti da li je broj pozitivan i negativan. Taj podatak se pamti u enumeraciji `enum pozitivnost {minus, plus}`. Ako je broj negativan, prvi znak je znak `'-'`, pa ga je za daljnji račun potrebno ga prebrisati pomicanjem svih znamenki jedno mjesto ulijevo.

Sljedeći korak je pretvaranje broja u binarni sustav za što je zaslužna funkcija `char *pretvori_10_u_binarno_string(char *broj)`. Koristeći se standardnim algoritmom za pretvaranje broja iz jedne baze u drugu, pretvore se posebno cijeli dio, posebno decimalni i na kraju spoje u jedan string. Funkcija za pretvaranje u binarni sustav poziva pomoćne funkcije za računanje sa stringovima (`char *pomnozi_decimalni_dio_s_2(char *decimalni_dio)`, `char *podijeli_cjelobrojno_s_2(char *cijeli_dio)`).

Da bi odredili normalizirani oblik broja, potrebno je prvo odrediti potenciju broja, odnosno indeks od kojeg počinje mantisa broja. To se može napraviti određivanjem mjesta decimalne točke te zaključivanjem kako se odnose te 3 varijable odnose jedna prema drugoj, što ovisi da li je početni broj veći ili manji od 1.

Sljedeće je potrebno kodirati potenciju broja, što radi funkcija `char *pretvori_potenciju_u_binarno(int potencija)`. Sada imamo sve podatke potrebne za upis u string `rjesenje1_IEEE`, pa to i napravimo. Zapisujemo bit za predznak, eksponent i mantisu. Ako mantisa ne stane, upisuju se prve 23 znamenke. Ovako dobiveno rješenje ispišemo fukcijom `void ispis_IEEE(char *IEEE)`, odredimo dekadski ekvivalent funkcijom `double iz_IEEE_u_10(char *IEEE, int potencija)`, te odredimo apsolutnu grešku funkcijom `char *oduzimanje_2_broja_iste_duljine(char *veci_broj, char *manji_broj)`. Zatim ponovimo postupak sa stringom `rjesenje2_IEEE`, koji dobijemo zbrajanjem jedinice sa prvim rješenjem (`char *dodaj_1_binarnom_broju(char *binarni_broj)`).

Na kraju usporedimo dvije vrijednoti apsolutnih grešaka (`int vrati_indeks_manjeg_broja_iste_velicine(char *broj1, char *broj2)`) te samo ispišemo broj čija je apsolutna greška manja.

4. IZVORNI KOD

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>

int vrati_indeks_manjeg_broja_iste_velicine(char *broj1, char *broj2)
{
	int i1 = strlen(broj1), i2 = strlen(broj2), n, i;

	i1 < i2 ? n = i1 : n = i2;

	/* Vrti broj dok se ne naiđe na znamenke koje su različite */
	for(i = 0; i < n; i++)
	{
		if(broj1[i] == '.') continue;

		if(broj1[i] > broj2[i])
			return 2;
		if(broj1[i] < broj2[i])
			return 1;
	}

	if(broj1[i] == NULL && broj2[i] == NULL)
		return 0;
	else if(broj1[i] == NULL)
		return 1;
	else
		return 2;


	return 1;
}

int valjanost_unesenog_broja(char *broj)
{
	int broj_tocaka = 0, i;

	/* Broj je nevaljan ako: */

	/* Počinje s znamenkama 00xxxx */

	/* Je jednak nuli */
	if(strlen(broj) == 1 && broj[0] == '0')
		return 0;

	/* Ako sadrži na prvom mjestu bilo koje druge znakove osim znamenaka i minusa
*/
	if((broj[0] > '9' || broj[0] < '0') && broj[0] != '-') return 0;
	if(broj[0] == '-' && (broj[1] > '9' || broj[1] < '0')) return 0;

	/* Počinje s znamenkama 00xxxx */
	if(broj[0] == '0' && broj[1] == '0') return 0;

	/* Ako sadrži više od jedne decimalne točke ili sadrži bilo koji znak koji nije
znamenka (ili točka) */
	for(i = 1; i < strlen(broj); i++)
	{
		if(broj[i] == '.')
			broj_tocaka++;
```

```
			else if(broj[i] > '9' || broj[i] < '0')
				return 0;
		}

		/* Ako sadrži više od jedne decimalne točke */
		if(broj_tocaka > 1)
			return 0;

		return 1;
}

char *iz_10_u_string(double broj)
{
		/* Deklaracija i inicijalizacija varijabli */
		int cijeli_dio = (int)(broj - (double)(broj - (int)broj)), brojac = 0, brojac2 =
0;
		double decimalni_dio = broj - cijeli_dio;
		char rjesenje[300];

		/* Zapis znamenaka cijelog dijela */
		do
		{
			rjesenje[brojac++] = cijeli_dio % 10 + '0';
			cijeli_dio /= 10;
		}while(cijeli_dio > 0);

		/* Završi zapis, okreni ga i dodaj na kraju točku */
		rjesenje[brojac] = NULL;
		strcpy(rjesenje, strrev(rjesenje));
		rjesenje[brojac++] = '.';

		/* Zapiši znamenke decimalnog dijela */
		do
		{
			decimalni_dio *= 10;
			rjesenje[brojac++] = (int)(decimalni_dio - (double)(decimalni_dio -
(int)decimalni_dio)) + '0';
			decimalni_dio = (double)(decimalni_dio - (int)decimalni_dio);
		}while(brojac2++ < 14);

		/* Kraj zapisa */
		rjesenje[brojac] = NULL;

		return rjesenje;
}

char *oduzimanje_2_broja_iste_duljine(char *veci_broj, char *manji_broj)
{
		/* Deklaracija i inicijalizacija varijabli */
		int ostatak1 = 0, ostatak2 = 0, i, n, znamenka1, znamenka2;
		char rezultat[300], pomocni_veci_broj[300], pomocni_manji_broj[350];

		strcpy(pomocni_veci_broj, veci_broj);
		strcpy(pomocni_manji_broj, manji_broj);

		/* Odredi koji broj ima manji broj decimala (pod pretpostavkom da imaju isti
broj znamenaka prije točke) */
		/* Broj koji ima manje, ako postoji, andopuni sa nulama do dužine dužeg broja */
		if(strlen(pomocni_veci_broj) < strlen(pomocni_manji_broj))
		{
			i = strlen(pomocni_manji_broj);
			n = strlen(pomocni_veci_broj);
```

```
			for(; i >= n; i--)
				pomocni_veci_broj[i] = '0';
		}
		else
		{
			i = strlen(pomocni_veci_broj);
			n = strlen(pomocni_manji_broj);

			for(; i >= n; i--)
				pomocni_manji_broj[i] = '0';
		}

		/*  */
		if(strlen(pomocni_veci_broj) < strlen(pomocni_manji_broj))
			i = strlen(pomocni_veci_broj);
		else
			i = strlen(pomocni_manji_broj);

		rezultat[i--] = NULL;

		/* Algoritam oduzimanja */
		for(; i >= 0; i--)
		{
			znamenka1 = pomocni_veci_broj[i] - '0';
			znamenka2 = pomocni_manji_broj[i] - '0' + ostatak1;

			if(znamenka1 + '0' == '.' || znamenka2 + '0' == '.') /* Ako se naiđe na
točku, zapiši je i u rješenju */
			{
				rezultat[i] = '.';
				continue;
			}

			if(znamenka1 - znamenka2 < 0)
			{
				znamenka1 += 10;
				ostatak2++;
			}

			rezultat[i] = znamenka1 - znamenka2 + '0';

			ostatak1 = ostatak2;
			ostatak2 = 0;
		}

		/* Ako su na početku dvije nule, prebriši jednu pomicanje ulijevo */
		while(rezultat[0] == '0' && rezultat[1] == '0')
		{
			for(i = 1; i <= strlen(rezultat); i++)
				rezultat[i - 1] = rezultat[i];
		}

		/* Ako su na kraju dvije nule, prebriši jednu pomicanjem kraja zapisa */
		while(rezultat[strlen(rezultat) - 1] == '0' && rezultat[strlen(rezultat) - 2] ==
'0')
			rezultat[strlen(rezultat) - 1] = NULL;

		return rezultat;
}

char *dodaj_1_binarnom_broju(char *binarni_broj)
```

```
{
	/* Inicijalizacija i deklaracija varijabli */
	char pomocni_binarni_broj[33];
	strcpy(pomocni_binarni_broj, binarni_broj);
	int ostatak = 1, i;

	/* Algoritam dodavanja jedinice (zbrajanja sa jedinicom) – dodavaj ostatak jedan
dok se ne naiđe na nulu */
	for(i = strlen(binarni_broj) – 1; ostatak; i--)
	{
		if(pomocni_binarni_broj[i] == '0')
		{
			pomocni_binarni_broj[i] = '1';
			ostatak = 0;
		}
		else
			pomocni_binarni_broj[i] = '0';
	}
	return pomocni_binarni_broj;
}

double potencija_2(int potencija)
{
	double rjesenje = 1;
	int i;

	/* Računaj potenciju broja 2 */
	if(potencija < 0)
		for(i = 0; i < -potencija; i++)
			rjesenje /= 2;
	else
		for(i = 0; i < potencija; i++)
			rjesenje *= 2;

	return rjesenje;
}

double iz_IEEE_u_10(char *IEEE, int potencija)
{
	/* Deklaracija i inicijalizacija varijabli */
	int potencije_2[30], pomocni_indeks = 1, i;
	double rjesenje = 0;

	/* Postavi vrijednost niza u kojem se pamte sve potencija broja 2 na nulu */
	memset(potencije_2, 0, sizeof(potencije_2));

	/* Zapiši u niz sve potencija broja 2 */
	for(i = 9; i <= 32; i++)
		if(IEEE[i] == '1')
			potencije_2[pomocni_indeks++] = -(i – 8);

	/* Zbroji sve potencije s eksponentom u normaliziranom zapisu broja te dobij
rezultat kao zbroj svih dobivenih potencija broja 2 */
	for(i = 0; i < pomocni_indeks; i++)
		rjesenje += potencija_2(potencije_2[i] + potencija);

	/* U slučaju negativnog broja */
	if(IEEE[0] == '1')
		rjesenje *= -1;

	/* Sigurni smo da konačno rješenje neće generirati grešku u zapisu jer se radi o
potencijama broja 2, odnosno konačnim zapisima u registru */
```

```
	return rjesenje;
}

void ispis_IEEE(char *IEEE)
{
	int i;

	/* Ispiši registar pomoću nula i jedinica, odvoji znakom ' | ' predznak,
eksponent i mantisu */
	printf(„%c“, IEEE[0]);

	printf(„ | „);

	for(i = 1; i < 9; i++)
		printf(„%c“, IEEE[i]);

	printf(„ | „);

	for(i = 9; i <= 32; i++)
		printf(„%c“, IEEE[i]);

	printf(„\n“);
}

char *pretvori_potenciju_u_binarno(int potencija)
{
	/* Deklaracija i inicijalizacija varijabli */
	char potencija_binarno[9], pomocno_binarno[9];
	int pomocni_indeks = 0, i;

	/* Algoritam dijeljenja s 2 radi pretvorbe u binarni sustav */
	do
	{
		pomocno_binarno[pomocni_indeks++] = potencija % 2 + '0';
		potencija /= 2;
	}while(potencija != 0 && pomocni_indeks < 8);

	/* Zavrsi zapis i okreni ga */
	pomocno_binarno[pomocni_indeks] = NULL;
	strcpy(pomocno_binarno, strrev(pomocno_binarno));

	/* Postavi sve znamenke rezultata na nulu */
	memset(potencija_binarno, '0', sizeof(potencija_binarno));

	/* Prepiši sve znamenke potencije u rezultat */
	for(i = 0; i < strlen(pomocno_binarno); i++)
		potencija_binarno[i + 8 – strlen(pomocno_binarno)] = pomocno_binarno[i];

	/* Kraj zapisa */
	potencija_binarno[8] = NULL;

	return potencija_binarno;
}

char *podijeli_cjelobrojno_s_2(char *cijeli_dio)
{
	/* Deklaracija i inicijalizacija varijabli */
	char pomocno_cijeli_dio[100];
	int i, ostatak = 0, znamenka;

	/* Algoritam dijeljenja s 2 */
	for(i = 0; i < strlen(cijeli_dio); i++)
```

```
	{
		znamenka = cijeli_dio[i] - '0' + ostatak * 10;
		ostatak = znamenka % 2;
		pomocno_cijeli_dio[i] = znamenka / 2 + '0';
	}
	pomocno_cijeli_dio[i] = NULL;

	/* Ako je prva znamenka jednaka nuli, prebrišimo je pomićići sve znamenke jedno
mjesto ulijevo */
	if(pomocno_cijeli_dio[0] == '0')
		for(i = 1; i <= strlen(pomocno_cijeli_dio); i++)
			pomocno_cijeli_dio[i - 1] = pomocno_cijeli_dio[i];

	return pomocno_cijeli_dio;
}

char *pomnozi_decimalni_dio_s_2(char *decimalni_dio)
{
	/* Deklaracija i inicijalizacija varijabli */
	char pomocno_decimalni_dio[100];
	int i, ostatak_1 = 0, ostatak_2, znamenka;

	/* Algoritam za množenje s 2 */
	for(i = strlen(decimalni_dio) - 1; i >= 0; i--)
	{
		znamenka = decimalni_dio[i] - '0';

		if(znamenka  >= 5)
			ostatak_2 = 1;
		else
			ostatak_2 = 0;
		pomocno_decimalni_dio[i] = (znamenka * 2) % 10 + '0' + ostatak_1;
		ostatak_1 = ostatak_2;
	}

	/* Ako je zadnja znamenka nula, prebrišimo je stavljanjem kraja */
	if(pomocno_decimalni_dio[strlen(decimalni_dio) - 1] == '0')
		pomocno_decimalni_dio[strlen(decimalni_dio) - 1] = NULL;
	else
		pomocno_decimalni_dio[strlen(decimalni_dio)] = NULL;

	return pomocno_decimalni_dio;
}

char *pretvori_u_binarno_cijeli_dio(char *cijeli_dio)
{
	/* Inicijalizacija i deklaracija varijabli */
	char binarno_cijeli_dio[100];
	int pomocni_indeks = 0, zadnja_znamenka;

	/* Algoritam pretvaranja u binarni sustav - dijeljenjem s 2 dok se ne dobije
nula */
	do
	{
		zadnja_znamenka = cijeli_dio[strlen(cijeli_dio) - 1] - '0';
		binarno_cijeli_dio[pomocni_indeks++] = zadnja_znamenka % 2 + '0';

		strcpy(cijeli_dio, podijeli_cjelobrojno_s_2(cijeli_dio));
	}while(strlen(cijeli_dio) != 0);

	/* Završi zapus i okreni dobiveni broj */
	binarno_cijeli_dio[pomocni_indeks] = NULL;
```

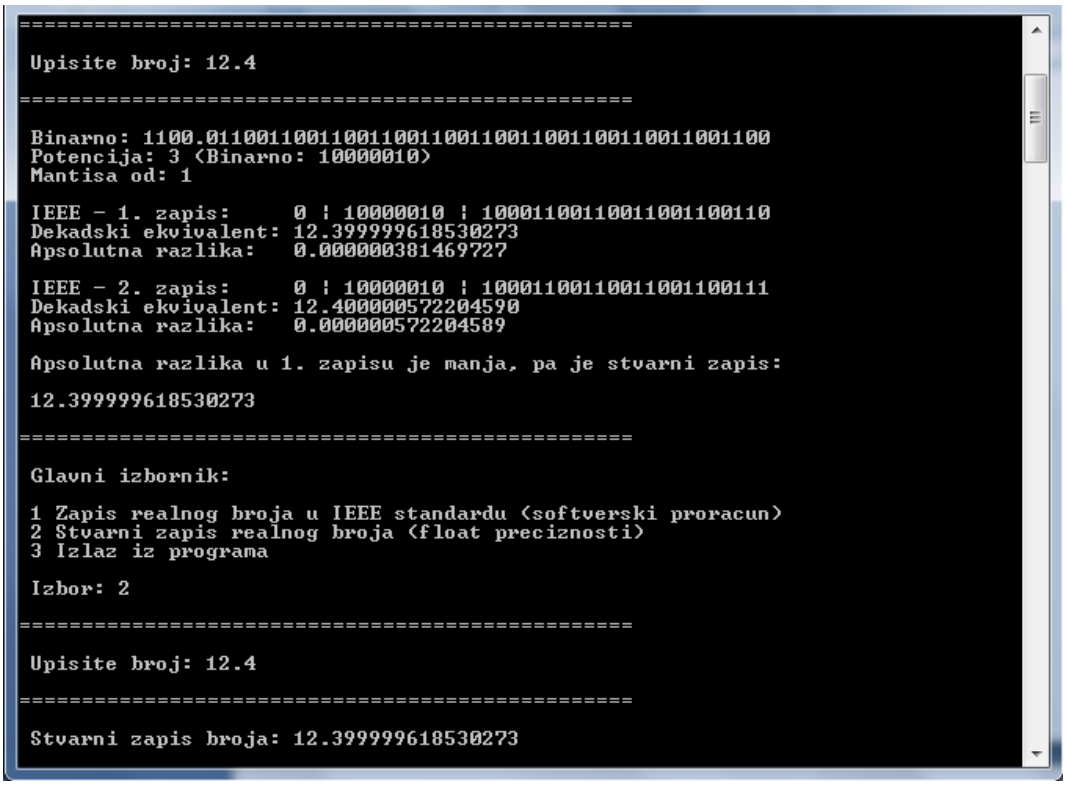
```
	strcpy(binarno_cijeli_dio, strrev(binarno_cijeli_dio));

	return binarno_cijeli_dio;
}

char *pretvori_u_binarno_decimalni_dio(char *decimalni_dio)
{
	/* Inicijalizacija i deklaracija varijabli */
	char binarno_decimalni_dio[300];
	int pomocni_indeks = 0, prva_znamenka;

	/* Algoritam pretvaranja u binarni sustav – množenjem sa 2 */
	do
	{
		prva_znamenka = decimalni_dio[0] - '0';

		if(prva_znamenka >= 5)
			binarno_decimalni_dio[pomocni_indeks++] = '1';
		else
			binarno_decimalni_dio[pomocni_indeks++] = '0';

		strcpy(decimalni_dio, pomnozi_decimalni_dio_s_2(decimalni_dio));
	}while(strlen(decimalni_dio) != 0 && pomocni_indeks < 270); /* Radi dok se
algoritam ne izvrši ili dok se ne zapiše 270 decimala */

	/* Kraj zapisa */
	binarno_decimalni_dio[pomocni_indeks] = NULL;

	return binarno_decimalni_dio;
}

char *pretvori_10_u_binarno_string(char *broj)
{
	/* Deklaracija i inicijalizacija varijabli */
	char cijeli_dio[100], decimalni_dio[100], binarno_cijeli_dio[100],
binarno_decimalni_dio[300], binarno[400];
	int pomocni_indeks = 0, i = 0;

	/* Odvoji cijeli dio – dio koji je do decimalne točke ili do kraja zapisa broja
*/
	for(i = 0; broj[i] != '.' && broj[i] != NULL; i++)
		cijeli_dio[i] = broj[i];
	cijeli_dio[i] = NULL;

	/* Ako ne postoji decimalni dio, stavi ga na nulu, u suprotnom odvoji ga */
	if(broj[i++] == NULL)
	{
		decimalni_dio[0] = '0';
		decimalni_dio[1] = NULL;
	}
	else
		do
		{
			decimalni_dio[pomocni_indeks++] = broj[i];
		}while(broj[i++] != NULL);

	/* Pretvori posebno cijeli dio, posebno decimalni u binarni sustav */
	strcpy(binarno_cijeli_dio, pretvori_u_binarno_cijeli_dio(cijeli_dio));
	strcpy(binarno_decimalni_dio, pretvori_u_binarno_decimalni_dio(decimalni_dio));

	/* U rješenje prvo zapiši cije dio */
	for(i = 0; i < strlen(binarno_cijeli_dio); i++)
```

```
			binarno[i] = binarno_cijeli_dio[i];
		binarno[i] = NULL;

		/* Ako je decimalni dio različit od nule, dodaj točku na kraj stringa te dodaj
sve decimale iza točke */
		if(binarno_decimalni_dio[0] != '0' || binarno_decimalni_dio[1] != NULL)
		{
			binarno[i++] = '.';
			for(pomocni_indeks = 0; pomocni_indeks <= strlen(binarno_decimalni_dio);
pomocni_indeks++)
				binarno[i + pomocni_indeks] =
binarno_decimalni_dio[pomocni_indeks];
		}

		return binarno;
}

void IEEE(char *broj)
{
		/* Deklaracija i inicijalizacija varijabli */
		char broj_binarno[400], binarno_potencija[9], rjesenje1_IEEE[33],
rjesenje2_IEEE[33];
		char razlika_rjesenja_1[300], razlika_rjesenja_2[300],
rjesenje1_IEEE_string[300], rjesenje2_IEEE_string[300];
		int potencija, pocetak_mantise, i, i2, manji_indeks;
		enum pozitivnost {minus, plus};
		double rjesenje1_IEEE_double, rjesenje2_IEEE_double;
		pozitivnost predznak;

		/* Ako je broj negativan, zapamti i pretvori ga i pozitivan */
		if(broj[0] == '-')
		{
			predznak = minus;
			for(i = 1; i <= strlen(broj); i++)
				broj[i - 1] = broj[i];
		}
		else /* U suprtonom, samo zapamti da je pozitivan */
			predznak = plus;

		/* Pretvori broj u binarni sustav */
		strcpy(broj_binarno, pretvori_10_u_binarno_string(broj));

		/* Računanje potencije broja u normaliziranom obliku */
		if(broj_binarno[0] == '1' && broj_binarno[1] == '.') /* Ako je oblika 1.xxx...
onda je potencija jednaka 0 */
			potencija = 0;
		else if(broj_binarno[0] == '0') /* Ako je broj oblika 0.xxxxx, onda je potencija
jednaka negativnom indeksu jedinice/kraja broja + 1 */
		{
			for(i = 2; broj_binarno[i] != '1' && broj_binarno[i] != NULL; i++) ;
				potencija = -(i - 1);
		}
		else /* Ako je oblika 1xxxx.xxxx, onda je potencija jednaka indeksu decimalne
tocke/kraja broja - 1 */
		{
			for(i = 1; broj_binarno[i] != '.' && broj_binarno[i] != NULL; i++) ;
				potencija = i - 1;
		}

		/* Računanje indeksa znamenke od koje se počinje zapisivati mantisa */
		if(potencija >= 1) /* Ako je potencija veća od 1, onda počni prepisivati mantisu
od druge znamenke */
```

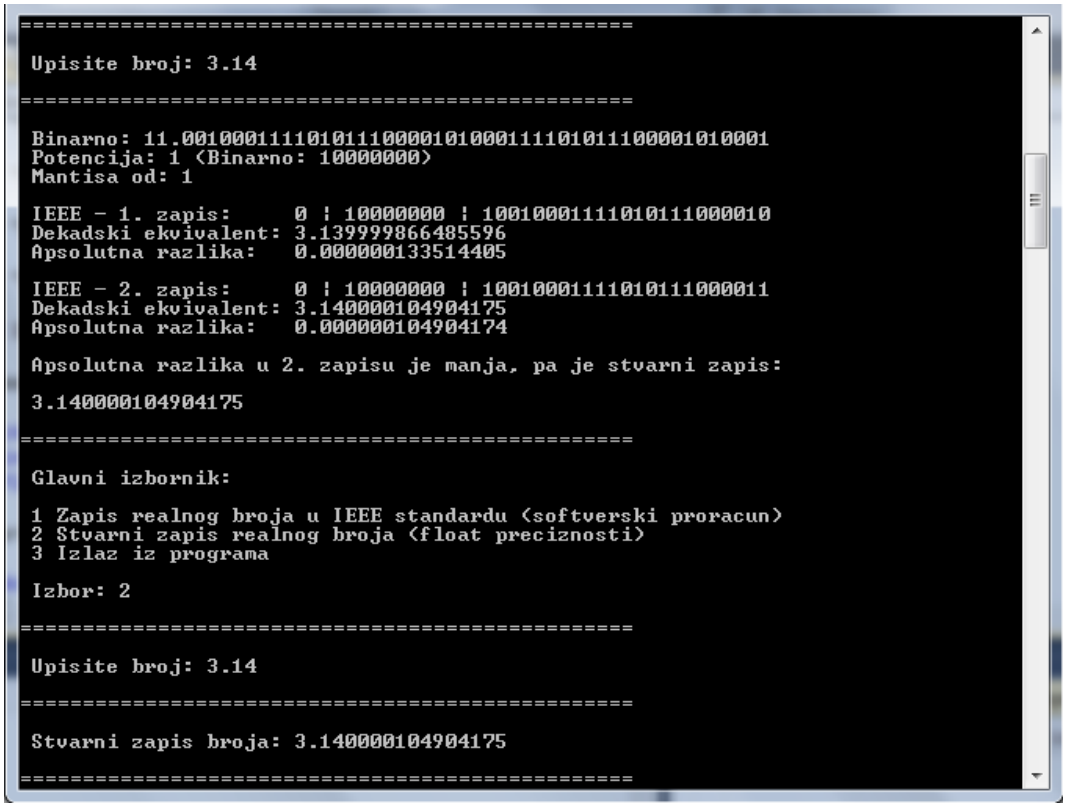
```
			pocetak_mantise = 1;
		else /* U suprtonom indeks pocetka mantise je jednak indeksu decimalne tocke + 3 ili 2 – potencija */
			pocetak_mantise = -potencija + 2;

		/* Za kodiranje potencije je potrebno dodati pomak od 127 */
		potencija += 127;

		/* Kodiranje potencije */
		strcpy(binarno_potencija, pretvori_potenciju_u_binarno(potencija));

		/* Ispis podataka */
		printf(„\n\n Binarno: %.50s\n Potencija: %d (Binarno: %s)\n Mantisa od: %d\n\n“, broj_binarno, potencija – 127, binarno_potencija, pocetak_mantise);

		/* Postavi sve znamenke rjesenja na nulu */
		memset(rjesenje1_IEEE, '0', sizeof(rjesenje1_IEEE));

		/* Bit za predznak */
		if(predznak == plus)
			rjesenje1_IEEE[0] = '0';
		else
			rjesenje1_IEEE[0] = '1';

		/* Zapis potencije */
		for(i = 0; i < 8; i++)
			rjesenje1_IEEE[i + 1] = binarno_potencija[i];

		/* Zapis mantise u rjesenje */
		i2 = 9;
		for(i = pocetak_mantise; i < strlen(broj_binarno) && i2 <= 32; i++) /* Zapisuj u rjesenje dok broj ne stane ili dok se registar ne popuni */
		{
			if(broj_binarno[i] == '.') continue; /* Ako se naiđe na decimalnu točku, preskoči ju */
			rjesenje1_IEEE[i2++] = broj_binarno[i];
		}
		rjesenje1_IEEE[32] = NULL; /* Kraj zapia */

		/* Ispis ovako dobivenog rjesenja (rjesenje 1) */
		printf(„ IEEE – 1. zapis:	„);
		ispis_IEEE(rjesenje1_IEEE);

		/* Pretvori ovakav zapis u realni broj i ispiši ga */
		rjesenje1_IEEE_double = iz_IEEE_u_10(rjesenje1_IEEE, potencija – 127);
		printf(„ Dekadski ekvivalent: %.15f\n“, rjesenje1_IEEE_double);

		/* Pretvori ovako dobiven broj u string da bi mogli odrediti apsolutnu razliku između ovog broja i početnog, ispiši tu razliku */
		if(rjesenje1_IEEE_double < 0)
			rjesenje1_IEEE_double *= -1;
		strcpy(rjesenje1_IEEE_string, iz_10_u_string(rjesenje1_IEEE_double));
		strcpy(razlika_rjesenja_1, oduzimanje_2_broja_iste_duljine(broj, rjesenje1_IEEE_string));
		printf(„ Apsolutna razlika:	%s\n\n“, razlika_rjesenja_1);

		/* Drugo moguće rješenje jest ono koje je najmanje moguće, a veće od prvog, odnosno potrebno je na mantisu probrojiti jedinicu  */
		strcpy(rjesenje2_IEEE, dodaj_1_binarnom_broju(rjesenje1_IEEE));

		/* Ispis ovako dobivenog rjesenja (rjesenje 1) */
		printf(„ IEEE – 2. zapis:	„);
```

```
	ispis_IEEE(rjesenje2_IEEE);

	/* Pretvori ovakav zapis u realni broj i ispiši ga */
	rjesenje2_IEEE_double = iz_IEEE_u_10(rjesenje2_IEEE, potencija - 127);
	printf(„ Dekadski ekvivalent: %.15f\n“, rjesenje2_IEEE_double);

	/* Pretvori ovako dobiven broj u string da bi mogli odrediti apsolutnu razliku
između ovog broja i početnog, ispiši tu razliku */
	if(rjesenje2_IEEE_double < 0)
		rjesenje2_IEEE_double *= -1;
	strcpy(rjesenje2_IEEE_string, iz_10_u_string(rjesenje2_IEEE_double));
	strcpy(razlika_rjesenja_2,
oduzimanje_2_broja_iste_duljine(rjesenje2_IEEE_string, broj));
	printf(„ Apsolutna razlika:   %s\n“, razlika_rjesenja_2);

	/* Odredi koji je broj bolji za zapis početnog broja tako da se usporede
apolutne razlike */
	manji_indeks = vrati_indeks_manjeg_broja_iste_velicine(razlika_rjesenja_1,
razlika_rjesenja_2);

	/* Ispis konačnog rješenja */
	switch(manji_indeks)
	{
		case 1:
			printf(„\n Apsolutna razlika u %d. zapisu je manja, pa je stvarni
zapis:\n\n %.15f\n“, manji_indeks, rjesenje1_IEEE_double);
			break;

		case 2:
			printf(„\n Apsolutna razlika u %d. zapisu je manja, pa je stvarni
zapis:\n\n %.15f\n“, manji_indeks, rjesenje2_IEEE_double);
			break;

		default:
			printf(„\n Apsolutne razlike su iste, pa je stvarni zapis:\n\n
%.15f\n“, rjesenje2_IEEE_double);
	}

}

int main (void)
{
	char broj[100];
	int pomocna_valjanost = 0, izbor;
	float float_broj;

	printf(„********************** FLOATING POINT - ERROR
**********************\n\n“);

	do
	{
		printf(„\n==================================================\n\n“);
		printf(„ Glavni izbornik:\n\n“);
		printf(„ 1 Zapis realnog broja u IEEE standardu (softverski
proracun)\n“);
		printf(„ 2 Stvarni zapis realnog broja (float preciznosti)\n“);
		printf(„ 3 Izlaz iz programa\n\n“);
		printf(„ Izbor: „);
		scanf(„%d“, &izbor);

		printf(„\n==================================================\n\n“);
```

```
		/* Glavni izbornik */
		switch(izbor)
		{
			case 1: /* Provjra zapisa realnog broja */
				printf(„ Upisite broj: „);
				scanf(„%s“, broj);

	printf(„\n================================================“);

				if(valjanost_unesenog_broja(broj))
				{
					pomocna_valjanost++;
					IEEE(broj);
				}
				else
					printf(„\n\n Krivo unseni broj, probajte
ponovo.\n“);
				break;

			case 2: /* Stvarni zapis realnog broja */
				printf(„ Upisite broj: „);
				scanf(„%f“, &float_broj);

	printf(„\n================================================“);

				printf(„\n\n Stvarni zapis broja: %.15f\n“, float_broj);

				break;

			case 3: /* Izlaz iz programa */
				printf(„ Izlaz iz programa...\n\n“);
				break;

			default: /* Krivi odabir */
				printf(„ Krivi odabir, molimo probajte ponovo.\n“);
				break;
		}

	}while(izbor != 3);

	system(„pause“);
	return 0;
}
```



Slika 2. – Ispis programa za broj 12.4



Slika 3. – Ispis programa za broj 3.14

5. ZAKLJUČAK

Velika većina brojeva se ne može zapisati točno u memoriji računala pa ih je potrebno zapisati uz što manju grešku. Prilikom zapisa realnih brojeva jednostruke preciznosti (tip `float` C-u), po IEEE standardu, radi se sljedeća provjera:

- Broj se zapiše u registar bez modifikacija. U slučaju da mantisa ne stane u registar, zapišu se 23 znamenke mantise
- Izračuna se apsolutna razlika između stvarnog i upisanog broja
- Na mantisu se binarno zbroji jedinica
- Izračuna se apsolutna razlika između stvarnog i ovako upisanog broja
- Prihvaća se onaj zapis čija je apsolutna razlika manja (manja pogreška zapisa)

Važno je napomenuti da računalo ne radi nužno točno ovakvu provjeru broja budući da u literaturi nisam našao dokaz za tu tvrdnju. Moguće je da se rezultat stvarne provjere samo poklapa s rezultatom ovakve metode koja je, kako pokazuje program, ispravna.

POPIS SLIKA

LITERATURA

Harris R.: „You're Going To Have To Think!“, Overload Journal #99; 2.12. 2011.; URL: http://accu.org/index.php/journals/1702

„IEEE Floating-Point Representation“ on MSDN Library; 2.12. 2011.; URL: http://msdn.microsoft.com/en-us/library/0b34tf65.aspx