Zapis podataka u računalu

Bit i bajt

Bajt (eng. *byte*) predstavlja oktet binarnih znamenaka, tj. 8 binarnih znamenaka. Suvremena računala rade s podatcima duljine 8, 16, 32, 64 bita, pa je bajt vrlo prikladna jedinica za skraćeno prikazivanje količine bitova. Često su u uporabi veće jedinice: kilobajt, megabajt, gigabajt, terabajt. Zbog naravi računala da se služi binarnim računanjem, jedinica pretvorbe nije 1000, već je $2^{10} = 1024$. To znači da 1 GB nije 1000 MB, već 1024 MB

Kôd i kôdiranje

ASCII

ASCII je kratica za American Standard Code for International Interchange. To je način kodiranja temeljen na engleskoj abecedi. Izrada je počela 1960. godine, objavljen je 1963. godine. Prvu veliku reviziju je doživio 1967. godine, a zadnja (konačna) verzija je objavljena 1986. godine.

Prva verzija je koristila 7 bitova, dakle, moglo se kodirati maksimalno $2^7 = 128$ znakova. Prva 32 znaka (0-31) su *non-printable*, tj. ne otiskuju se na ekranu. Ti znakovi su služili kao nadzorno-upravljački znakovi (npr. zvučni signal, novi red pisača, ...). Daljnjih 95 znakova (32-127) su slova (engleska abeceda), brojevi, znakovi interpunkcije. Poslijednji (128) znak je ESC.

Ubrzo se uvidjelo da treba stvoriti sličan kôd koji bi omogućavao kodiranje i specijalnih znakova ostalih jezika (npr. u hrvatskom jeziku č, đ...u francuskom ô, é). Rješenje ovog problema se vidjelo u proširenju ASCII kôda. Naime, kako se registri obično sastoje od 8 bitova, a osnovna inačica ASCII kôda koristi samo 7 (i to nižih!) bitova, jednostavnom računicom dolazimo do slijedećeg podatka - ako bi za kodiranje koristili svih 8 bitova, mogli bi kodirati maksimalno $2^8 = 256$ znakova. Tako je stvoren Extended ASCII kôd. Prvih 128 znakova je preuzeto iz osnovnog ASCII, a preostalih 128 je služilo za kodiranje specifičnih znakova nekih jezika. Hrvatska inačina proširenog ASCII kôda se zove CROSCII.

Međutim, dogodilo se da je pojedini jezik imao više inačica ASCII kôda, što je ponekad otežavalo kodiranje jer se isti znak nije uvijek kodirao istim brojem. O rješenju tog problema pogledajte slijedeće potpoglavlje!

USASCII code chart

Bits b_7 b_6 b_5 →	0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4 ↓ b_3 ↓ b_2 ↓ b_1 ↓ Row ↓ Column →	0	1	2	3	4	5	6	7
0 0 0 0	0 NUL	DLE	SP	0	@	P	`	p
0 0 0 1	1 SOH	DC1	!	1	A	Q	a	q
0 0 1 0	2 STX	DC2	"	2	B	R	b	r
0 0 1 1	3 ETX	DC3	#	3	C	S	c	s
0 1 0 0	4 EOT	DC4	$	4	D	T	d	t
0 1 0 1	5 ENQ	NAK	%	5	E	U	e	u
0 1 1 0	6 ACK	SYN	&	6	F	V	f	v
0 1 1 1	7 BEL	ETB	'	7	G	W	g	w
1 0 0 0	8 BS	CAN	(	8	H	X	h	x
1 0 0 1	9 HT	EM	)	9	I	Y	i	y
1 0 1 0	10 LF	SUB	*	:	J	Z	j	z
1 0 1 1	11 VT	ESC	+	;	K	[	k	{
1 1 0 0	12 FF	FS	,	<	L	\	l	|
1 1 0 1	13 CR	GS	-	=	M	]	m	}
1 1 1 0	14 SO	RS	.	>	N	^	n	~
1 1 1 1	15 SI	US	/	?	O	_	o	DEL

Slika 1: ASCII kôd

Kôdovi u tablici su poredani u stupce i retke. U stupcima su poredani po četiri viša bita (b_7, b_6, b_5), a u retcima po četiri niža bita (b_4, b_3, b_2, b_1). Važno je primjetiti da nije naveden bit b_8. U osnovnom ASCII kôdu uzimamo da je $b_8 = 0$.

Često se pojedini znak iz tablice prikazuje heksadekadskim kôdom, a ne binarnim. Razlog tome je što jedna znamenka heksadekadskog sustava zamjenjuje četiri znamenke binarnog sustava, pa je prikaz jednostavniji.

Npr. slovo "D" možemo prikazati kao 01000100, ali i kao 44. Ponekad se naglašava da je riječ o heksadekadskom sustavu na način da iza broja dopišemo slovo "h" (44h), kako ne bi bilo zabune (neki izvore navode kodove u dekadskom sustavu, u tom slučaju se iza broja nadopiše slovo "d").

Unicode

Problem sa ASCII kôdom je taj što različite zemlje koriste različite kôdne stranice. Čak i unutar jedne zemlje možemo naići na više kôdnih stranica. Time je komunikacija i razmjena informacija otežana jer nije svaki znak kôdiran istim brojem (osim prvih 128 znakova - oni su standardizirani za SVE kôdne stranice!). Rješenje tog problema se vidi u razvijanju i korištenju Unicode.

Unicode koristi jedinstven broj za svaki znak, bez obzira na platformu, bez obzira na program, bez obzira na jezik. Njime je opisan SVAKI znak iz SVIH poznatih živućih (pa čak i nekih "mrtvih") jezika. Time nema "opasnosti" da će dokument koji napišemo u Hrvatskoj biti drugačije prikazan na nekom računalu u Australiji.

Unicode obično koristi 2 bajta, tj. 16 bitova, čime je moguće kodirati čak $2^{16} = 65536$ različitih znakova.

Kôdne točke Unicoda standarda se mogu podijeliti na različite načine, najčešće na 17 *ravnina*, od kojih svaka ima 65536 kôdnih točaka. Prva ravnina (ravnina 0) služi za kodiranje velike većine znakova koji se koriste u modernim jezicima. Naziva se *Basic Multilingual Plane (BMP)* - Osnovna Multijezična Ravnina.

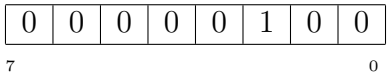
Prikaz brojeva u računalu

Prikaz cijelih brojeva u računalu

Brojeve u računalu prikazujemo binarnim brojem. Pretvorba brojeva se vrši na način opisan u poglavlju Brojevni sustavi. Za primjere ćemo koristiti 8 bitne registre. Tako broj moramo nadopuniti nulama sa lijeve strane kako bi dobili prikaz u registri.

Primjer: Broj $4_{10} = 100_2$

U registru će taj broj biti zapisan na slijedeći način:



Međutim, pitanje je kako ćemo prikazati negativne brojeve? Osmi bit registra (onaj sa najvećom težinskom vrijednošću!) određuje predznak. Ako je on 1, broj je negativan, a ako je 0, broj je pozitivan. Postoje tri različita načina prikaza cijelih brojeva.

I. način - prikaz predznakom i veličinom

Kod ovog se načina odvojeno manipulira predznakom i veličinom. U praktičnom računanju to znači slijedeće: nižih sedam bitova određuje broj, a osmi bit određuje samo predznak.

Primjer:

$36_{10} = 00100100_2$ $\qquad\qquad$ $-36_{10} = 10100100_2$

Izvođenje računskih operacija je relativno kompliciranо ukoliko koristimo ovaj način zapisa.

Najmanji broj koji je moguće prikazati ovim načinom je $11111111_2 = -127_{10}$, a najveći mogući je $01111111_2 = +127_{10}$.

Osim kompliciranog računanja, kod ovog načina nailazimo na još jedan problem: problem negativne nule. Naime, nulu možemo zapisati na dva načina: $00000000_2 = -0_{10}$ i $10000000_2 = +0_{10}$.

II. način - prikaz predznakom i komplementom

Kod ovog načina postupamo na slijedeći način: ukoliko je najviši bit 1, ostalih sedam komplementiramo.

Primjer:

$36_{10} = 00100100_2$ $\qquad\qquad$ $-36_{10} = 11011011_2$

I kod ovog načina je problem relativno kompliciranog računanja.

Najmanji broj koji je moguće prikazati ovim načinom je $10000000_2 = -127_{10}$, a najveći mogući je $01111111_2 = +127_{10}$.

Ponovno nailazimo na problem negativne nule. Nulu moćemo zapisati na dva načina: $11111111_2 = -010$ i $00000000_2 = +0_{10}$.

III. način - prikaz predznakom i dvojnim komplementom

Kod ovoga načina pozitivne brojeve prikazujemo najvišim bitom 0 (predznak +!) i veličinom, a negativne brojeve najvišim bitom 1 (predznak -!) i dvojnim komplementom.

Jednostavniji način pretvorbe broja u dvojni komplement je slijedeći: počevši od najmanjeg značajnog bita invertiramo sve bitove nakon prve 1.

Primjer: $36_{10} = 00100100_2$ $\qquad\qquad$ $-36_{10} = 11011100_2$

Najmanji broj koji je moguće prikazati ovim načinom je $10000000_2 = -128_{10}$, a najveći mogući je $01111111_2 = +127_{10}$.

Kod ovog načina nema problema negativne nule - ona je jedinstvena.

Koji je način bolji?

	I. način	II. način	III. način
Računske operacije	Komplicirano	Komplicirano	Jednostavno
Raspon (za 8 bitova)	[-127, 127]	[-127, 127]	[-128, 127]
Problem -0	Postoji	Postoji	Ne postoji

Iz svega navedenog možemo zaključiti da je najpraktičnije negativne brojeve prikazivati dvojnim komplemente!

Prikaz racionalnih i realnih brojeva u računalu

Racionalni i realni brojevi se u računalu mogu zapisati na dva načina: sa pomičnim zarezom ili sa nepomičnim zarezom.

Prikaz nepomičnim zarezom nije praktičan jer koristi fiksan dio registra za prikaz cijelog broja i fiksan dio za prikaz decimalnog dijela broja. Zbog toga je mogući raspon brojeva koje možemo prikazati tim načinom poprilično uzak, i ovisi o veličini registra.

U široj je uporabi prikaz broja sa pomičnim zarezom koji je definiran standardom IEEE 754. Dva su načina prikaza broja sa pomičnim zarezom - jednostrukom preciznošću (koji koristi 32 bita) i dvostrukom preciznošću (koji koristi 64 bita).

Prikaz jednostrukom preciznošću

Za ovaj se prikaz koriste 32 bita.

Najviši bit koristi za prikaz predznaka. Ako je on 1 broj je negativan, a ako je 0 broj je pozitivan.

Daljnjih 8 bitova služi za prikaz karakteristike, K.

Zadnjih 23 bita služi za prikaz mantise, M.

Kako bi broj prikazali u pogodnom obliku, moramo ga normalizirati. Normaliziranim brojem nazivamo brojeve koji imaju zapis $1.M_2 \cdot 2^{be}$.

Npr. broj $101.01_2 = 1.0101_2 \cdot 2^2$

Kako se normalizacijom broja postiže oblik $\mathbf{1.}M$, vodeću jedinicu ne pamtimo u računalu (podrazumijeva se!), i nazivamo je *skriveni bit*.

Broj M nazivamo *mantisa*, a be nazivamo *binarni eksponent*. *Karakteristiku* određujemo tako da binarni eksponent zbrojimo sa 127 (time izbjegavamo prikaz negativnih eksponenata!) i taj rezultat preračunamo u binarni sustav; $K_2 = (be + 127)_{10}$.

Najmanji broj koji je moguće prikazati ovim načinom je $1.401298464324817 \cdot 10^{-45}$, a najveći $3.402823669209 \cdot 10^{38}$.

Specijalni slučajevi kod ovog prikaza su slijedeći:

1. Ako je $K = 0$ i svi bitovi mantise su 0, onda se radi o broju 0
2. Ako je $K = 255$ i svi bitovi mantise su 0, onda se radi o $-\infty$ (ako je $P = 1$), odnosno o $+\infty$ (ako je $P = 0$)
3. Ako je $K = 255$, a neki bit mantise **nije** 0, onda se radi o NaN (Not a Number = Nije Broj)

Primjer 1:

Prikažimo broj 5.75_{10}

$5.75_{10} = 101.11_2 = 1.0111_2 \cdot 2^2$

$P = 0$, jer je broj pozitivan

$be = 2$, $K = 2 + 127 = 129_{10} = 10000001_2$

$M = 0111 = \{$nadopunimo nulama kako bi imali 23 bita!$\} = 01110000000000000000000$

Konačno, naš broj u registru ima zapis:

0	10000001	01110000000000000000000

Primjer 2:

Koji broj u registru ima zapis

1	10000011	11010010000000000000000

?

$K = 10000011_2 = 131_{10}$; $be = 131 - 127 = 4$

$M = 11010010000000000000000_2 = 1101001_2$

$1.1101001 \cdot 2^4 = 11101.001_2 = 29.125_{10}$

ALI...$P = 1$, pa se radi o broju -29.125.

Prikaz dvostrukom prezicnošću

Za ovaj se prikaz koriste 64 bita.

Najviši bit koristi za prikaz predznaka. Ako je on 1 broj je negativan, a ako je 0 broj je pozitivan.

Daljnjih 11 bitova služi za prikaz karakteristike, K.

Zadnjih 52 bita služi za prikaz mantise, M.

Normalizaciju brojeva kod dvostruke preciznosti radimo na analogan način kao i kod jednostruke preciznosti. Razlika kod dvostruke preciznosti je što je $K = be + 1023$.

Najmanji broj koji je moguće prikazati ovim zapisom je $4.9406 \cdot 10^{-324}$, a najveći $1.797693134862316 \cdot 10^{308}$.

Specijalni slučajevi kod ovog prikaza su slijedeći:

1. Ako je $K = 0$ i svi bitovi mantise su 0, onda se radi o broju 0
2. Ako je $K = 2047$ i svi bitovi mantise su 0, onda se radi o $-\infty$ (ako je $P = 1$), odnosno o $+\infty$ (ako je $P = 0$)
3. Ako je $K = 2047$, a neki bit mantise **nije** 0, onda se radi o NaN (Not a Number = Nije Broj)

Zadatci za vježbu

1. Preračunajte:

1. 50 KB u bitove
2. 12 MB u kilobajte
3. 120 GB u megabajte

2. Kodirajte slijedeći niz:

1. 49 6e 66 6f 52 4d 61 74 49 4b 61
2. 00110010 00101011 00110010 00111101 00110100
3. Danas je utorak!

3. Prikažite slijedeće brojeve u registru (NAPOMENA: za prikaz negativnih brojeva se služite dvojnim komplementom):

1. 98_{10}
2. -56_{10}
3. -122_{10}

4. Koji su brojevi prikazani u registru slijedećim zapisom? (NAPOMENA: za prikaz negativnih brojeva se koristi prikaz dvojnim komplementom)

1. 10101100_2
2. 01001001_2
3. 11111000_2

5. Prikaži slijedeće brojeve u registru (NAPOMENA: koristiti jednostruku preciznost!):

1. 25.125_{10}
2. -65.5_{10}
3. -12.25_{10}

6. Koji su brojevi u registru prikazani slijedećim zapisom?

1. | 0 | 10000101 | 00011111000000000000000 |
2. | 1 | 10000111 | 10001111011000000000000 |
3. | 1 | 01111110 | 10101000000000000000000 |