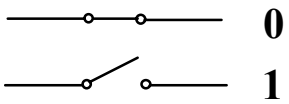
1. ŠTEVILSKI SISTEMI

1.1 Številske enote

1.1.1 Bit

Obdelava podatkov v mikroprocesorski tehniki sloni na *dvojiškem* (*binarnem*) številskem sistemu. Takšen sistem je izbran zato, ker ga je tehnično najlažje realizirati. Dve ločeni in edini možni stanji zelo enostavno ustvarimo s stikalnim elementom (slika 1. 1):

- stikalo zaprto (kontakta sklenjena, tok teče),
- stikalo odprto (kontakta razklenjena, tok ne teče).



Slika 1. 1: Določanje dveh stanj s pomočjo stikala

V praksi so stikala realizirana s polprevodniškimi elementi (tranzistorji in diodami) ter pasivnimi elementi (upori in kondenzatorji). Sodobni stikalni elementi so navadno izdelani v MOS (npr. HCMOS) tehnologiji. Uporabljena tehnologija nanašanja različnih polprevodniških plasti omogoča njihovo fizično realizacijo na izredno majhni površini (npr. nekaj stotisoč tranzistorjev na čipu), kjer se debeline plasti merijo v mikronih (10^{-6} m). Na ta način je možno na enoto površine spraviti veliko število podobnih stikal, ki preklapljajo z izredno visoko frekvenco (nekaj deset ali celo nekaj sto MHz) in imajo izredno majhno porabo (npr. nekaj sto μW v mirujočem, "standby" režimu).

Dve stanji s slike 1. 1 lahko ponazorimo s številoma 0 in 1 (ali "L" in "H": angl. low in high). Kateremu izmed njiju bomo dodelili posamezno številsko vrednost, je stvar dogovora. S tem smo določili najmanjšo enoto odločanja ("DA" in "NE", ali "RESNIČNO" in "NERESNIČNO" - angl. "TRUE" in "FALSE"). Za zapis obeh možnosti potrebujemo le eno dvojiško številko - *digit* (latinska beseda za prst). Od tod tudi ime najmanjšega nosilca digitalne informacije: *bit* (angl. **bi**nary digi**t**)[1].

V praksi realiziramo binarna stanja z napetostnimi ali tokovnimi signali različnih nivojev. Ti so posledica preklopnih stanj tranzistorja. Z ozirom na tehnološko realizacijo mikroelektronskega vezja ter njegovo konkretno aplikacijo razlikujemo več standardov, ki določajo višino nivoja za obe stanji (slika 1. 2). Najbolj razširjen je TTL standard, ki določa nazivno napetost 0 V za signal 0 in 5 V za signal 1. V praksi je tako strogo definirane nivoje nemogoče doseči zaradi motenj in napetostnih padcev, zato sta za obe stanji predvidena

[1] Tukaj imamo opravka z besedno igro, saj *bit* v angleščini pomeni tudi košček, malenkost.

pasova (tabela 1.1). Če zaradi kateregakoli razloga napetosti ali tokovi zasedejo vmesne vrednosti, logično stanje ostaja nedoločeno.

Slika 1. 2: Pasova za določanje nivojev logičnih stanj 0 in 1

Tehnologija	U_{IH} **[V]**	U_{IL} **[V]**	U_{OH} **[V]**	U_{OL} **[V]**
Standardna TTL	> 2,0	< 0,8	> 2,4	< 0,4
S-TTL	> 2,0	< 0,8	> 2,7	< 0,5
LS-TTL	> 2,0	< 0,8	> 2,7	< 0,5
CMOS	4,0 - 8,0	1,0 - 2,0	4,5 - 9	< 0,5
HCMOS (U_S = 4,5 V)	> 0,7 · U_S	< 0,3 · U_S	> (U_S - 0,5V)	< 0,4

Pojasnilo:
U_{IH}: vhodna napetost v logično vezje, logični nivo 1
U_{IL}: vhodna napetost v logično vezje, logični nivo 0
U_{OH}: izhodna napetost iz logičnega vezja, logični nivo 1
U_{OL}: izhodna napetost iz logičnega vezja, logični nivo 0

Tabela 1.1: Logični nivoji za različne logične družine [3]

V programirljivih krmilnikih oz. industrijskih procesorjih se pogostokrat uporabljata nivoja 0 V in 24 V za napetostne signale ali 0 mA in 20 mA za tokovne signale (npr. pri krmilnikih Simatic tvrdke SIEMENS [4] itd.

Omenimo še to, da dodelitev višje napetosti ali toka signalu 1 velja za t.i. "pozitivno logiko". Pri "negativni logiki" ustreza višji napetosti signal 0.

1.1.2 Nibble

Število različnih stanj, ki jih zapišemo z enim samim bitom, je omejeno na le dve. Zato je smiselno bite povezati v večje enote (tipe, formate) in na ta način povečati število možnih stanj[2]. Podatek, ki sestoji iz dveh bitov, lahko doseže štiri možna stanja, tribitni pa osem (tabela 1.2).

[2] Podobno je pri decimalnih številih: ena števka (enice) omogoča le 10 različnih stanj (od 0 do 9), dve števki (desetice in enice) 100 = 10^2 (0 - 99), tri (stotice, desetice in enice) 1000 = 10^3 (0 - 999) itd.

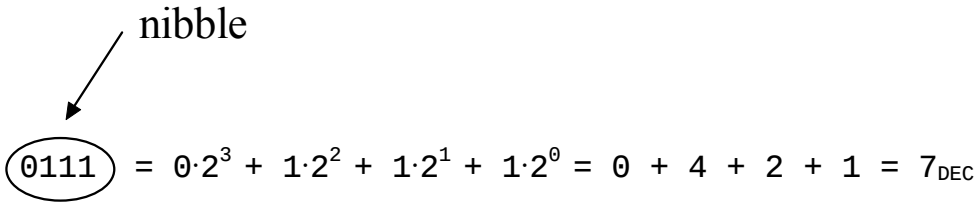
1 bit	2 bita	3 biti
0	00	000
1	01	001
	10	010
	11	011
		100
		101
		110
		111

Tabela 1.2: Enobitne, dvobitne in tribitne enote

Sedaj že lahko govorimo o *ločljivosti* (angl. resolution), ki je določena z izrazom $1/2^n$ kjer je n število bitov. Zaenkrat se bomo ustavili pri formatu štirih bitov ($2^4 = 16$), ki ga imenujemo *nibble* (dobesedno angl. grižljaj)[3].

Binarnemu zapisu lahko dodelimo tudi kvantitativen pomen z *binarno* ali *dvojiško kodo* (angl. binary code; indeks BIN). Baza tega sistema je 2 analogno *decimalni* ali *desetiški kodi* (angl. decimal; indeks DEC)[4], z bazo 10, ki jo uporabljamo v vsakdanjem življenju.

Na primeru nibbla s slike 1. 4 vidimo način interpretacije binarnega podatka v decimalni kodi. Skrajnji desni bit ima utež 1 (2^0), naslednji 2 (2^1), največjo pa ima skrajnji levi bit (v tem primeru $2^3 = 8$). Z dodajanjem bitov na levi se povečuje njihova utež in s tem tudi maksimalna vrednost celotnega števila. Bita z najmanjšo utežjo (skrajnje desni) in največjo utežjo (skrajnji levi) označujemo - ne glede na število bitov v podatku - s kraticami *LSB* (angl. Least Significant Bit - najmanj pomemben bit) in *MSB* (angl. Most Significant Bit - najbolj pomemben bit)[5].



Slika 1. 3: Pretvorba iz binarne v decimalno kodo

Maksimalno število, ki ga zapišemo v enem nibblu, je $1111_{BIN} = 15_{DEC}$. Splošna formula za izračun maksimalnega števila je:

[3] Nibble je le nekakšna neformalna podatkovna enota med bitom in zlogom, ki nam služi pri pretvorbi iz binarne v heksadecimalno kodo, zato tega izraza ponavadi ne uporabljamo.

[4] V nadaljnjem tekstu bomo posamezne formate označevali z ustreznimi indeksi. Decimalna števila bomo pisali z indeksom DEC ali pa brez kakršnegakoli indeksa.

[5] Pri označevanju posameznih bitov znotraj nekega formata izhajamo iz njihove uteži: **LSB** označujemo vedno kot ničti bit (npr. LSB naslovnega vodila je **A0**), **MSB** pa je označen z zaporedno številko n-1 (npr. MSB 16-bitnega naslovnega vodila je **A15**). Posebej je treba biti pozoren na možne napake zaradi besedne igre: prvi bit ima indeks 0, šestnajsti pa 15 (v 16-bitnem formatu)!

$$N_{max} = 2^n - 1, \qquad (1.\ 1)$$

kjer je n število bitov formata. Navidezno neskladje med ločljivostjo ($2^4 = 16_{DEC}$) in maksimalnim številom (15_{DEC}) pojasnimo s tem, da je manjkajoče šestnajsto veljavno število 0.

Zapis števil v binarni kodi se izkaže kot zelo nepregleden, zlasti ko imamo opravka s 16- ali 32-bitnimi enotami. Zato takšna števila raje interpretiramo s *šestnajstiško* ali *heksadecimalno kodo* (angl. hexadecimal; kratica HEX), kjer je baza število 16. Že prej smo ugotovili, da lahko z enim nibblom ustvarimo 16 različnih kombinacij (števil). Iz tega sledi, da štiri bite v nibblu nadomestimo z eno šestnajstiško številko. Šestnajstiške številke od 0 do 9 so ekvivalentne decimalnim, nadaljnje številke (10_{DEC} - 15_{DEC}) pa označujemo s črkami[6] a_{HEX}, b_{HEX}, c_{HEX}, d_{HEX}, e_{HEX}, f_{HEX} (glej tudi tabelo 1.3)!

1.1.3 Byte (zlog)

Z dodajanjem bitov pridemo do zelo pomembne enote, ki je sastavljena iz osmih bitov: *byte* (bajt ali zlog). Njegova ločljivost je $2^8 = 256$, kar pomeni, da je LSB $1/256_{DEC}$ celotnega obsega. Zlog se je udomačil kot osnovna enota, ki jo uporabljamo v mikroprocesorski tehniki (vse višje standardizirane enote so mnogokratniki zloga). MSB (sedmi bit) pri zlogu ima utež 2^7.

Tabela 1.3 kaže različne možne kombinacije enega byta ter njihove interpretacije v šestnajstiški in desetiški kodi[7]. Pri vseh načinih kodiranja je pomembno poudariti, da so kode le različni načini interpretacije istega binarnega števila!

Vzemimo binarno število $1100\ 0110_{BIN}$. Najlaže ga pretvorimo v šestnajstiško število: $c6_{HEX}$ ($1100_{BIN} = c_{HEX}$, $0110_{BIN} = 6_{HEX}$). Decimalni zapis lahko dobimo bodisi iz binarnega

$$1100\ 0110_{BIN} = (1\cdot2^7 + 1\cdot2^6 + 1\cdot2^2 + 1\cdot2^1)_{DEC} = (128 + 64 + 4 + 2)_{DEC} = \\ = 198_{DEC},$$

ali iz šestnajstiškega zapisa

$$c6_{HEX} = (12\cdot16^1 + 6\cdot16^0)_{DEC} = (192 + 6)_{DEC} = 198_{DEC}.$$

Iz tabele 1.3 in enačbe (1. 1) je razvidno, da je maksimalno decimalno število, ki ga lahko zapišemo z enim zlogom, 255_{DEC} ($11111111_{BIN} = ff_{HEX}$).

V tabeli je prikazana tudi t.i. *BCD koda* (angl. Binary Coded Decimal - dvojiško kodirana decimalna števila). Za razliko od ostalih omenjenih kod, kjer vsak levo dodani bit dobi povečano utež, velja ta logika pri BCD kodi le znotraj enega nibbla, in sicer le za njegove decimalne ekvivalente.

[6] Načeloma lahko te številke šestnajstiških števil zapišemo bodisi z velikimi ali malimi črkami.

[7] Na sliki sta v binarnem zapisu nibbla ločena le zaradi lažje interpretacije. Pri normalnem zapisu binarnega števila tega presledka ni.

BIN binary (*binarno*)	**DEC** decimal (*decimalno*)	**HEX** hexadecimal (*heksadecimalno*)	**BCD** Binary Coded Decimal system (*binarno zakodirani decimalni sistem*)
0000 0000	0	00	0000 0000
0000 0001	1	01	0000 0001
0000 0010	2	02	0000 0010
0000 0011	3	03	0000 0011
0000 0100	4	04	0000 0100
0000 0101	5	05	0000 0101
0000 0110	6	06	0000 0110
0000 0111	7	07	0000 0111
0000 1000	8	08	0000 1000
0000 1001	9	09	0000 1001
0000 1010	10	0a	0001 0000
0000 1011	11	0b	0001 0001
0000 1100	12	0c	0001 0010
0000 1101	13	0d	0001 0011
0000 1110	14	0e	0001 0100
0000 1111	15	0f	0001 0101
0001 0000	16	10	0001 0110
...	...	...	...
0111 1111	127	7f	0001 0010 0111
1000 0000	128	80	0001 0010 1000
...	...	...	...
1111 1111	255	ff	0010 0101 0101

Tabela 1.3: Števila v zlogu v dvojiški, desetiški, šestnajstiški in BCD kodi

Primer:

Pri pretvorbi števila

$1100\ 0110_{BIN} = 198_{DEC}$

v BCD izhajamo iz njegove decimalne kode. Vsaki decimalni števki bomo priredili binarno četverico (nibble).

$1_{DEC} = 0001_{BCD},\ 9_{DEC} = 1001_{BCD},\ 8_{DEC} = 1000_{BCD}$ oz.

$1100\ 0110_{BIN} = 198_{DEC} = 1\ 1001\ 1000_{BCD}$.

BCD kodo bi v decimalnem zapisu lahko interpretirali na naslednji način:

$(1 \cdot 2^0) \cdot 10^2 + (1 \cdot 2^3 + 1 \cdot 2^0) \cdot 10^1 + (1 \cdot 2^3) \cdot 10^0 = 1 \cdot 10^2 + 9 \cdot 10^1 + 8 \cdot 10^0$.

Kot vidimo, imamo v enem zapisu opravkahkrati z desetiško in binarno bazo.

Pazi!

Zapisa v binarni in BCD kodi sta si zelo podobna, saj so številke le enice in ničle, njuna interpretacija pa je popolnoma različna.

Do sedaj smo omenili le najpogosteje uporabljane kode. Nekatere druge kode, kot je npr. Excess-3 BCD koda, se niso širše uveljavile.

Zlog je že dolgo časa referenčna enota, npr. za določanje kapacitete pomnilnika z mnogokratnikom zlogov: Kbyte (izgovarjava: kilobajt), Mbyte (megabajt), Gbyte (gigabajt)[8]. Pri tem moramo biti zelo previdni saj te predpone ne označujejo potence števila 10,

$1\ K = 10^3 = 1000,\ 1\ M = 10^6 = 1\ 000\ 000,\ 1\ G = 10^9 = 1\ 000\ 000\ 000,$

pač pa izhajajo iz potence števila 2:

$1\ K = 2^{10} = 1024_{DEC},\ 1\ M = (2^{10})^2 = 1.048.576_{\ DEC},\ 1\ G = (2^{10})^3 = 1.073.741.824_{\ DEC}.$

Torej 64 Kbyte spomina ni 64000 bytov, temveč 64·1024 = 65536 bytov!

Preden nadaljujemo z ostalimi enotami, omenimo še kodo, ki jo sicer uporabljamo za prikaz znakov. *ASCII* (kratica od "American Standard Code for Information Interchange") je 7-bitna koda, ki se pogosto uporablja v računalnikih in komunikacijah. Z njo kodiramo razne vidne znake (črke, številke, ločila...), kakor tudi znake, ki krmilijo izpis ali vsebujejo nekatere dodatne informacije (npr. koda $0d_{HEX}$ označuje CR, angl. carriage return, ki ga uporabljamo za potrdilo vnosa).

1.1.4 Beseda

Povečanje ločljivosti z dodajanjem bitov nas pripelje do naslednjega značilne enote, *besede* (angl. word), ki je sestavljena iz dveh zlogov, torej šestnajstih bitov. Maksimalno število, ki ga lahko zapišemo v eni besedi je (glej tudi (1. 2), pri $n = 16$):

$1111\ 1111\ 1111\ 1111_{BIN} = ffff_{HEX} = 65535_{DEC} = 110\ 0101\ 0101\ 0011\ 0101_{BCD}.$

Ločljivost 16-bitne enote je $1/2^{16} = 1/65536$.

1.1.5 Dvojna in štirikratna beseda

Večina najsodobnejših procesorjev (npr. CPU32 v MC68332, ki ga bomo analizirali v nadaljevanju) sloni na 32-bitni enoti ali *dolgi* oz. *dvojni besedi* (angl. long word ali double word). Največje število v dvojiški kodi bo zapisano s dvaintridesetimi enkami, kar bo v šestnajstiškem zapisu ffff $ffff_{HEX}$. Decimalni ekvivalent je 4 294 967 295.

[8] To neskladje naj bi rešile nove SI enote: kibi, mebi, gibi [34].

Naslednja enota je dvakrat večja (64 bitov) in jo imenujemo *štirikratna beseda* (angl. quad word). V večini primerov se javlja le kot rezultat množenja dveh dvojnih besed ali kot deljenec v nekaterih operacijah.

Na koncu še enkrat povzemimo vse do sedaj omenjene enote v njihovem izvirnem zapisu:

1 byte = 8 bit,
1 word = 2 byte = 16 bit,
1 long word = 2 word = 4 byte = 32 bit,
1 quad word = 2 long word = 4 word = 8 byte = 64 bit.

1.2 Nepredznačena in predznačena cela števila

S povečanjem dolžine enot dosežemo večjo ločljivost in maksimalno možno številsko vrednost (npr. 65535 pri 16-bitni enoti). Pri tem smo imeli opravka le s *celimi* (angl. integer) *nepredznačenimi* (angl. unsigned) števili. V tem podpoglavju bomo govorili o načinu ponazoritve in računanja s predznačenimi celimi števili ter o zapisu števila s plavajočo vejico.

Obstaja več možnih načinov za zapis *negativnih* oz. *predznačenih* števil (angl. signed), najbolj razširjena pa je uporaba *dvojiškega komplementa* (angl. two's complement).

1.2.1 Izračun dvojiškega komplementa

Postopek formiranja dvojiškega komplementa ali negativne vrednosti nekega števila je sestavljen iz dveh korakov:

1. Formiranje *eniškega komplementa* (angl. one's complement): ustvarimo ga tako, da ničle v bitnem vzorcu zamenjamo z enkami in obratno.
2. Prištevanje števila 1 k eniškemu komplementu.

Zgornji postopek si bomo ogledali na izračunu nasprotne vrednosti (pogovorno: negiranju) števila 4_{DEC}, ki smo ga zapisali v 16-bitni enoti.

4_{DEC} = 0000 0000 0000 0100$_{BIN}$

1. korak: 1111 1111 1111 1011$_{BIN}$
2. korak: +1$_{BIN}$
rezultat: 1111 1111 1111 1100$_{BIN}$

Pravilnost postopka lahko preverimo na dva načina:

- Negiranje negativne vrednosti mora rezultirati z enako vrednostjo pozitivnega predznaka:

  -4_{DEC} = 1111 1111 1111 1100$_{BIN}$

  1. korak: 0000 0000 0000 0011$_{BIN}$
  2. korak: +1$_{BIN}$

rezultat: $0000\ 0000\ 0000\ 0100_{BIN} = +4_{DEC}$

- Rezultat seštevanja števila z lastno nasprotno vrednostjo je nič:

$$\begin{array}{ll} & 1111\ 1111\ 1111\ 1100_{BIN} = -4_{DEC} \\ & 0000\ 0000\ 0000\ 0100_{BIN} = +4_{DEC} \\ \hline (1)\ & 0000\ 0000\ 0000\ 0000_{BIN} = 0_{DEC} \end{array}$$

prenos

Zaradi prenosa enice kot posledice seštevanja (od 3. bita naprej do MSB) pride do njenega "izpada" izven 16-bitnega obsega. Tukaj nas ne zanima njena usoda, temveč preostali biti v besedi. Lahko omenimo le to, da se pri izvajanju te operacije v mikroprocesorju ta enica "shrani" v bit prenosa (angl. carry bit; oznaka C - glej tudi pogl. 4.1.2).

1.2.2 Prikaz negativnih števil v različnih formatih

Čeprav smo z dvojiškim komplementom rešili problem ponazoritve negativnih števil, narekuje njegova uporaba nekatera pravila in omejitve.

Z uvajanjem zapisa negativnih števil s pomočjo dvojiškega komplementa nismo spremenili ločljivosti posamezneenote (zloga, besede ali dvojne besede), ki je bila prej celotno na voljo le pozitivnim številom (npr. 0 - 65535_{DEC} za 16-bitno enoto). Posledica tega je delitev celotnega obsega na negativno in pozitivno področje.

Oglejmo si to na primeru 8-bitne enote (slika 1. 5), pri vseh višjih pa velja podobno načelo. Brez upoštevanja dvojiškega komplementa je obseg pozitivnih vrednosti:

$0000\ 0000_{BIN}$ - $1111\ 1111_{BIN}$ 00_{HEX} - ff_{HEX} 0_{DEC} - 255_{DEC}.

Z uvajanjem dvojiškega komplementa se je celotno področje razdelilo na dva dela:

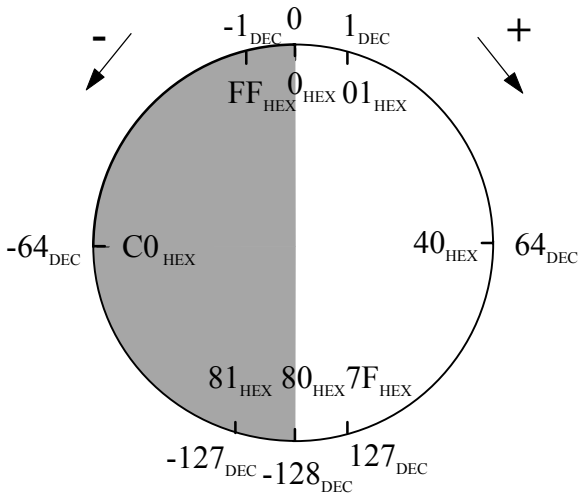
[-128_{DEC} ... -1] in [0_{DEC} ... $+127_{DEC}$].

Kot vidimo, je po maksimalni absolutni vrednosti negativno število večje od maksimalnega pozitivnega. Razlog je v tem, da med pozitivno polovico področja ($2^8/2 = 256/2 = 128$) štejemo tudi ničlo! Nasprotna vrednost števila -128_{DEC} ($1000\ 0000_{BIN}$) v tej enoti ne obstaja, kar lahko dokažemo s pomočjo dvojiškega komplementa:

$$\begin{array}{ll} \text{eniški komplement:} & 0111\ 1111_{BIN} \\ & \underline{\qquad\quad +1_{BIN}} \\ \text{rezultat:} & 1000\ 0000_{BIN} = -128_{DEC}. \end{array}$$

Torej smo dobili enako vrednost kot pred negiranjem.

Zaradi razlike med interpretiranjem binarne kode v nepredznačenih in predznačenih (dvojiški komplement) številih bomo prve označevali s standardizirano oznako U (angl. unsigned), druge pa s S (angl. signed).



Slika 1. 4: Prikaz pozitivnih in negativnih števil v 8-bitnem formatu

Zelo zanimivo si je ogledati dvojiške in šestnajstiške ekvivalente nekaterih pozitivnih in negativnih števil po logiki dvojiškega ekvivalenta (slika 1. 5 in tabela 1.4). V skrajnjem desnem stolpcu so ekvivalenti binarnim zapisom v U notaciji.

BIN	HEX	DEC (S)	DEC (U)
1000 0000	80	-128	128
1000 0001	81	-127	129
1000 0010	82	-126	130
1111 1110	fe	-2	254
1111 1111	ff	-1	255
0000 0000	00	0	0
0000 0001	01	1	1
0000 0010	02	2	2
0111 1110	7e	126	126
0111 1111	7f	127	127

Tabela 1.4: Nekatere vrednosti pozitivnih in negativnih števil v 8-bitnem formatu v različnih kodah

Takoj ugotovimo, da binarne oz. heksadecimalne kode za števila od 0 - 127_{DEC} v S in U notaciji enako interpretiramo. Do razlik pride pri negativnih S številih. Najmanjše (po absolutni vrednosti) negativno število po S notaciji (-1_{DEC}) je enako največjemu številu po U notaciji (255). Večanje negativnih S števil je analogno manjšanju vednosti v U notaciji, kar nazorno prikažemo s krogom na sliki 1. 5.

Iz povedanega lahko ugotovimo nekaj dejstev, ki veljajo tudi za S števila v besednih in dvobesednih enotah:

- pri največjem pozitivnem številu je MSB enak nič, vsi ostali biti pa so enaki ena: ($7fff_{HEX}$ = 32767_{DEC} oz. 7fff ffff$_{HEX}$ = 2 147 483 647_{DEC}),
- pri negativnem številu z največjo absolutno vrednostjo je MSB enak ena, vsi ostali biti pa so enaki nič (8000_{HEX} = -32768_{DEC} oz. 8000 0000$_{HEX}$ = -2 147 483 648_{DEC}),
- negativno število z najmanjšo absolutno vrednostjo (-1_{DEC}) je sestavljeno iz samih enic (ffff$_{HEX}$ oz. ffff ffff$_{HEX}$).

MSB je pri S notaciji vedno indikacija predznaka (1 za negativno, 0 za pozitivno število), za ugotovitev absolutne vrednosti pa moramo izračunati dvojiški komplement.

V tabeli 1.5sta še enkrat zbrana oba zapisa v pomembnejših formatih in dveh kodah.

Formati	Predznačeno (S)		Nepredznačeno (U)	
	DEC	HEX	DEC	HEX
Byte	[-128 ... 127]	[80 ... 7F]	[0 ... 255]	[0 ... FF]
Word	[-32768 ... 32767]	[8000 ... 7FFF]	[0 ... 65535]	[0 ... FFFF]
Long word	[$\pm 2{,}147 \times 10^9$]	[80000000 ... 7FFFFFFF]	[0 ... $4{,}295 \times 10^9$]	[0 ... FFFFFFFF]

Tabela 1.5: Povzetek U in S zapisov za različne enote

Dvojiškemu komplementu smo namenili veliko prostora zato, ker ga pogosto srečamo pri delu s procesorji, zlasti ko programiramo v zbirniku ali ko interpretiramo vrednosti bipolarnih A/D in D/A pretvornikov. V nadaljevanju si bomo ogledali primera možnih pasti pri računanju s predznačenimi števili.

1.2.2.1 Presežek

Presežek (angl. overflow) je ena pogostih posledic izvajanja aritmetičnih operacij v procesorju in lahko pripelje do napačne interpretacije rezultatov. Presežek je posledica nezmožnosti zapisa rezultata operacije med dvema veljavnima številoma v enoti operandov. Posredno je v to "vpletena" tudi logika dvojiškega komplementa.

Vzemimo za primer seštevanje dveh 16-bitnih negativnih števil -30000_{DEC} in -8000_{DEC}. Pričakovani rezultat je -38000_{DEC}. Iz tabele 1.6 je razvidno, da tega rezultata v 16-bitni enoti ne moremo zapisati. Oglejmo si operacijo na nivoju binarnih števil (dvojiška komplementa smo že izračunali):

```
  1000 1010 1101 0000BIN        8ad0HEX    -30000DEC
+ 1110 0000 1100 0000BIN       +e0c0HEX   +(-8000DEC)
① 0110 1011 1001 0000BIN        6b90HEX     27536DEC
└► prenos
```

Tudi brez podrobne analize rezultata izračuna lahko ugotovimo, da smo kot rezultat seštevanja dveh negativnih števil dobili pozitivno število, saj je MSB rezultata enak nič. Torej je rezultat napačen, kar lahko ugotovimo iz njegove decimalne interpretacije. Ta pa ni takšna, kot bi jo pričakovali.

Zaradi presežka se postavi poseben bit v statusnem registru (običajno označen z V ali OV), lahko pa se generira ustrezna prekinitev (glej pogl. 4.1.2).

1.2.2.2 Prenos števil v višjo enoto

Pri prenosu števil iz nižje enote (npr. zloga) v višjo enoto (npr. besedo) je tudi treba upoštevati logiko dvojiškega komplementa. V bistvu gre tukaj za pravilno preslikavo MSB. Oglejmo si primer prenosa dveh 8-bitnih števil v 16-bitno enoto.

- **Pozitivno število**
  Pri prenosu moramo paziti na to, da se vsi "dodani" biti na levi strani zapolnijo z ničlami (število v oklepajih označuje enoto zapisa):

  $75_{HEX}(8)$ → $0075_{HEX}(16)$ ali
  $0111\ 0101_{BIN}(8)$ → $0000\ 0000\ 0111\ 0101_{BIN}(16)$.

- **Negativno število**
  Pri prenosu se vsi dodani dodani biti zapolnijo z enicami:

  $85_{HEX}(8)$ → $ff85_{HEX}(16)$ ali
  $1111\ 0101_{BIN}(8)$ → $1111\ 1111\ 1000\ 0101_{BIN}(16)$.

Nekaj podobnega se dogaja tudi pri operacijah aritmetičnega pomika (angl. arithmetic shift) v levo ali desno (glej tudi pogl. 4.2.4).

Zaradi predznaka na primer pri hardverski vezavi 12-bitnega bipolarnega A/D (ali D/A) pretvornika na 16-bitno podatkovno vodilo poravnamo MSB naprave in MSB vodila (štirje odvečni skrajnje desni biti vodila pri tem ne igrajo nobene vloge; glej tudi pogl. 10.3.4.4.1).

Osnovno pravilo pri vseh opisanih operacijah je torej prenos MSB iz nižje v višjo enoto oz. njegovo ohranjanje.

1.3 Števila s plavajočo vejico

Aritmetika večine procesorjev, ki se uporabljajo v krmilnih in regulacijskih nalogah sloni na uporabi celih števil. Ta omejitev zahteva od programerja dodaten trud pri iskanju optimalnega izkoriščanja številskega obsega. Zato je veliko enostavneje delati z *realnimi števili*.

Slednje v mikroračunalnikih najpogosteje zapisujemo v obliki števil s *plavajočo vejico* (angl. floating point) [12]. Veljavni ANSI/IEEE standard 754-1985 za aritmetiko binarnih števil s plavajočo vejico je bil sprejet leta 1985. Na tem projektu je več kot deset let delalo 92 strokovnjakov iz vrst matematikov, računalničarjev in ostalih inženirjev z univerz in iz industrije. Razlog za tako obsežno delo je bila zmeda na področju zapisa realnih števil, saj so nekateri proizvajalci uporabljali binarne ali decimalne zapise, v (bivši) Sovjetski zvezi pa celo *trinarne* (baza 3). Celo med binarnimi računalniki so obstajale inačice, ki so za bazo imele število 2, 8 ali 16.

V tem poglavju bomo opisali t.i. *format z dvojno natančnostjo* (angl. double precision format). Poleg tega obstajata še format z *enojno natančnostjo* (angl. single precision) ter format s *razširjeno natančnostjo* (angl. extended precision).

Števila s plavajočo vejico so povečinoma normirana. To pomeni, da jih lahko zapišemo v obliki

$$x = \pm(1+f)\cdot 2^{e},$$

kjer f *mantisa* ali *ulomek* (angl. mantissa, fraction), e pa *eksponent*. Mantisa mora zadostiti pogoju normiranja (glej tudi poglavje 10):

$$0 \le f < 1.$$

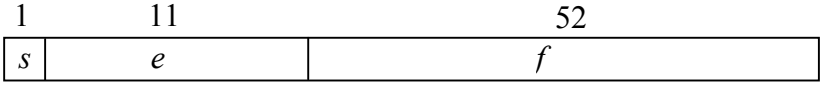
Mantiso zapišemo v 52-bitnem formatu. To pomeni, da mora biti $2^{52}\cdot f$ celo število v območju

$$0 \le 2^{52} f < 2^{53}. \qquad (1.3)$$

Območje eksponenta e je:

$$-1022 \le e \le 1023. \qquad (1.4)$$

Realna števila s plavajočo vejico in z dvojno natančnostjo zapisujemo v 64-bitnem formatu:



Bit s določa predznak (angl. sign) števila, eksponent je lahko pozitiven (zelo velika števila) ali negativen (zelo majhna števila), mantisa pa nepredznačena.

Za eksponent je rezerviranih enajst bitov, torej je njegova ločljivost 2^{11} = 2048. Iz podpoglavja o dvojiškem komplementu celih števil je razvidno, da je interval (pozitivna in negativna števila) za ta format [-1024,+1023]. Zastavlja se vprašenje, zakaj dve največji negativni števili (-1024 in -1023) nista zajeta v intervalu eksponenta (1. 3)? Odgovor leži v zapisu ekstremnih primerov, ko:

- števila presegajo dovoljeno področje zapisa ali
- jih sploh ne moremo zapisati kot število.

V prvem primeru je takšen rezultat t.i. *neskončno število* - *Inf* (angl. infinity). Takrat je $f = 0$ in $e = 1024$, pri tem pa morajo veljati tudi relacije kot so: 1/Inf = 0, Inf + Inf = Inf....

Včasih se pa rezultata aritmetične operacije sploh ne da zapisati v obliki realnega števila. Te vrednosti označujemo kot *NaN* (angl. Not-a-Number: “ni veljavno število”) in so posledica operacij, kot so 0/0 ali Inf - Inf.