Assembly programozás: 2. gyakorlat

Számrendszerek:

Kettes (bináris) számrendszer: {0, 1}
Nyolcas (oktális) számrendszer: {0,..., 7}
Tízes (decimális) számrendszer: {0, 1, 2, ... , 9}
16-os (hexadecimális számrendszer): {0, 1, 2, ... , 9, A, B, C, D, E, F}

Alaki érték: 0, 1, 2, ..., 9, ...
Helyi érték: attól függ, hogy a szám melyik pozíción áll

Példa:

$583_{10} = 5 * 100 + 8 * 10 + 3 * 1 = 5 * 10^2 + 8 * 10^1 + 3 * 10^0$
$583_{16} = 5 * 256 + 8 * 16 + 3 * 1 = 5 * 16^2 + 8 * 16^1 + 3 * 16^0$

Számrendszerek közötti konverzió:

$583_{16} = ?_{10}$

256	16	1	$5 * 256 + 8 * 16 + 3 * 1 = 1411_{10}$
5	8	3	

$583_{10} = ?_{16}$
$16^3 = 4096$, $16^2 = 256$, $16^1 = 16$, $16^0 = 1$

583_{10}	/	256	=	2
71_{10}	/	16	=	4
7_{10}	/	1	=	7

$583_{10} = 247_{16}$

$583_{10} = ?_2$
$2^{10} = 1024$, $2^9 = 512$, $2^8 = 256$, $2^7 = 128$, $2^6 = 64$, $2^5 = 32$, $2^4 = 16$, $2^3 = 8$, $2^2 = 4$, $2^1 = 2$, $2^0=1$

583_{10}	/	512	=	1
71_{10}	/	256	=	0
71_{10}	/	128	=	0
71_{10}	/	64	=	1
7_{10}	/	32	=	0
7_{10}	/	16	=	0
7_{10}	/	8	=	0
7_{10}	/	4	=	1
3_{10}	/	2	=	1
1_{10}	/	1	=	1

$583_{10} = 1001000111_2$

Algoritmikus megoldás:

Ha decimális számrendszerből binárisba váltunk át, akkor a decimális számot mindig kettővel kell osztani egészen addig, amíg a hányadosként 1-et nem kapunk. Az egyes osztások után feljegyezzük a maradékot. A decimális szám bináris számrendszerbeli alakját úgy kapjuk, hogy a maradékokat visszafelé egymás után írjuk. A visszafelé olvasást az indokolja, hogy mire 1-et kaptunk hányadosként, addig n-szer osztottunk le 2-vel, így 2^n lesz az a legnagyobb 2-hatvány, amellyel a szám osztható. Visszafelé haladva a többi bináris számjegy is meghatározott. A visszafelé olvasás tulajdonképpen azt jelenti, hogy az addigi bináris számot egy helyiértékkel balra toljuk (szorozzuk 2-vel) és hozzáadjuk a maradékot, amely a legkisebb helyiértékre kerül, a 0-nak 0, az 1-nek pedig 1 lesz az értéke.

$583_{10} = ?_2$		$583_{10} = ?_{16}$	
/2	Maradék	/16	Maradék
583_{10}	1	583	7
291_{10}	1	36	4
145_{10}	1	2	2
72_{10}	0		
36_{10}	0		
18_{10}	0		
9_{10}	1		
4_{10}	0		
2_{10}	0		
1_{10}	1		
$583_{10} = 1001000111_2$		$583_{10} = 247_{16}$	

$110010_2 = ?_{10}$

64	32	16	8	4	2	1
	1	1	0	0	1	0

$110010_2 = 32 + 16 + 2 = 50$

Bináris-hexadeximális konverzió:

A bináris számjegyeket a legkisebb helyiértékű számtól 4-essével konvertáljuk. Ha a számjegyek száma nem osztható 4-gyel, akkor legnagyobb helyértékű számjegyeket 0-val pótoljuk.

Példa:

1010110100010010 = 1010 1101 0001 0010 = AD12

Hexadecimális-bináris konverziónál a hexadecimális számjegyeket 4 bináris számjeggyé alakítjuk.

Törtek konvertálása

A törtszámok konvertálásánál a számot egészrészre és törtrészre bontjuk fel. Vegyük itt is a decimális-bináris konverziót! Az egészrészt ugyanúgy váltjuk át, ahogy az egészszámokat az előző algoritmussal. A törtrész átváltásánál pedig mindig meg kell szorozni az aktuális törtrészt a bináris számrendszer alapjával (2-vel), és az egészrészeket kell feljegyezni. A egészrészeket egymás után összeolvasva kapjuk a törtrész bináris változatát. Az algoritmus akkor áll meg, ha a törtrész 0 lesz. Elképzelhető, hogy véges decimális szám törtrésze binárisan nem lesz véges.

$35.132_{10} = ?_2$

/2	Maradék	egész	*2
			0.132
35	1	0	.264
17	1	0	.528
8	0	1	.056
4	0	0	.112
2	0	0	.224
1	1	0	.448
		0	.896
		1	.792
		1	.584
		...	...

Nem biztos, hogy véges tizedes tört binárisan is véges lesz!

Horner elrendezés:

Hogy néz ki a 287_{10} szám Horner elrendezésben?
(((2 * 10 + 8) * 10 + 7)

Hogy néz ki a 572_8 szám Horner elrendezésben?
(((5 * 8 + 7) * 8 + 2)

Összeadás:

Mi a 11101101_2 és a 000111_2 számok összege?

```
  11101101
+  000111
  11110100
```

Mi a $14AA5_{16}$ és a $F32_{16}$ számok összege?

```
    14AA5
+    F 32
   15 9D7
```

<u>Előjeles fixpontos számok ábrázolásai:</u>

<u>Előjeles abszolút érték:</u> balról az első bit az előjel: 0, ha a szám pozitív, 1, ha negatív

- a 0 kétféleképpen ábrázolható: 10000000, 00000000
- a legkisebb szám -127, a legnagyobb 127

<u>Egyes komplemens:</u> az első bit az előjel (0, ha pozitív; 1, ha negatív)
A szám (-1)-szerese úgy kapható meg, hogy minden bitjét ellenkezőjére állítjuk.

- a 0 kétféleképpen ábrázolható: 00000000, 11111111

<u>Kettes komplemens:</u> az első bit az előjel, 0: pozitív, 1: negatív
egy negatív szám úgy kapható meg, hogy az abszolút értékének egyes komplemenshez hozzáadunk 1-et.

- a legkisebb szám a -128, a legnagyobb a 127
- a nulla egyértelműen ábrázolható
  Pl.: 127 = 01111111, 0 = 00000000, -128 = 10000000, -1 = 11111111

Feladat: számoljuk ki a 43_{10} kettes komplemensét.
Megoldás:
$43_{10} = 00101011_2$ 00101011 (természetes számoknál nincs változás)

Feladat: számoljuk ki a -123_{10} kettes komplemensét.
Megoldás: a gyakorlaton
$123_{10} = 01111011$
10000100 (ez egyes komplemens)
$-123_{10} = 10000101$ (ez már kettes komplemens)

Többletes számábrázolás:
a szám és a többlet összegét ábrázoljuk előjel nélkül, *m* bites szám esetén a többlet 2^{m-1} vagy $2^{m-1}-1$ (az alábbi példák 128-többletes értékeket mutatnak be)

pl.:	$35_{10} = 00100011$	többletes érték: 10100011_2	35 + 128
	$-35_{10} = 10100011$	többletes érték: 01011101_2	- 35 + 128 = 93
	128 = 10000000,	többletes érték: 00000000	-128 + 128 = 0

- a legkisebb szám -128, a legnagyobb 127,
- a nulla egyértelműen meghatározható
- a 2^{m-1} többletes ábrázolás azonos a kettes komplemenssel fordított előjellel.
- lebegőpontos számok kitevő-részénél használják

Lebegőpontos számok:
Általában normalizált alakban ábrázoljuk. A nulla általában csupa 0-ból áll, meghatározható a legkisebb és legnagyobb ábrázolható szám, valamint a legkisebb és legnagyobb hiba.
Az IEEE 754 standard szerint
single: 32 bit: előjel (1 bit), kitevőrész (8 bit 127-többletes), a törtrész abszolút értéke (23 bit)
double: 64 bit: előjel (1 bit), kitevőrész (11 bit 1023-többletes), a törtrész abszolút értéke (52 bit)

Normalizált számok:
kitevőrész = kitevő + 127 (-126, ..., 127). Közvetlenül a törtrész elé kell képzelni egy 1-est (implicit bit) és a bináris pontot.
pl.: $1 = 001111111000...0000_2$ $1 = 2^0$

előjel kitevőrész 1. törtrész
(pozitív) (0+127=127) (2, implicit)

$0.5 = 001111110000...0000_2$ $0.5 = 2^{-1}$

előjel kitevőrész 1. törtrész
(pozitív) (-1+127=126)

Normalizálatlan számok:
ha a kitevőrész = 0, ilyenkor a kitevő = -126 nem 127
a bináris pontot implicit 0 előzi meg, ami nincs ábrázolva
pl.: $2^{-127} = 000000000100...0000_2$
$-2^{-149} = 100000000000...0001_2$
Ha a kitevőrész = 255, akkor vagy túl nagy számokról, vagy NaN-ról beszélünk.

Feladatok:

1. Váltsd át 01110101 számot hexadecimális számrendszerbe!

2. Váltsd át a A564 hexadecimális számot bináris számmá!

3. Add össze a B592 és E12 hexadecimális számokat!

4. Hogyan ábrázolnád a 6.45 decimális számot egyszeres (single) pontosságú, normalizált lebegőpontos számként?