**Laborator 1 - Suport teoretic**

**Conversii şi operaţii în diferite baze de numeraţie**

***Se va studia:***

* conversia numerelor întregi şi zecimale din baza 10 într-o bază oarecare, în special baza 16 şi 2
* conversia inversă, dintr-o bază oarecare în baza 10, mai ales din baza 16 şi 2 în baza 10.
* conversia din baza 16 direct în baza 2 şi invers.

***Consideraţii teoretice***

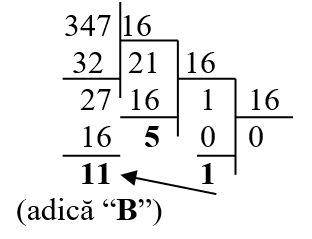
Un sistem de numeraţie este constituit din totalitatea regulilor de reprezentare a numerelor cu ajutorul anumitor simboluri denumite cifre. Pentru orice sistem de numeraţie , numărul semnelor distincte pentru cifrele sistemului este egal cu baza (b). Deci pentru baza b=2 (numere scrise în binar) semnele vor fi cifrele 0 şi 1. Pentru baza b=16 (hexazecimal) semnele vor fi 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F. Se observă că pentru numerele scrise într-o bază mai mare decât baza 10 (zecimal) se folosesc şi alte simboluri (litere) pe lângă cifrele obişnuite din baza 10. Astfel, în cazul numerelor scrise în hexazecimal, literele A,B,C,D,E,F au ca şi valori asociate 10,11,12,13,14,15. Moduri de notare: scrierea la sfârşitul numărului în paranteză a bazei, de exemplu: 100101001(2), sau 17A6B(16).

***Conversia numerelor din baza 10 într-o bază oarecare***

Algoritmul cel mai simplu constă în împărţirea succesivă a numărului scris în baza 10 la baza spre care se doreşte conversia (se împarte numărul la bază, iar în continuare se împarte câtul obţinut la bază ş.a.m.d. până când câtul devine 0), după care se iau resturile obţinute în ordine inversă, care constituie valoarea numărului în baza cerută.

***Exemple:***

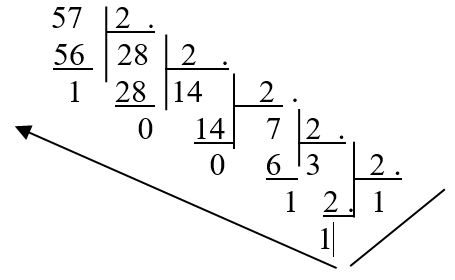
1. Să se convertească 347 din baza 10 în baza 16(H) Mai întâi se converteşte în baza 16 pentru că aceasta se realizează prin mai puţine împărţiri decât conversia în baza 2.



Astfel, luând resturile în ordine inversă obţinem 15B(H).

347(D) = 15B(H)

1. Să se convertească 57 din baza 10 în baza 2(B).



57(D) = 111001(B)

Există şi o metodă mai rapidă de conversie a numerelor între bazele 2 şi 16 ţinând cont că pentru fiecare cifră hexa există 4 cifre binare corespondente:

|  |  |  |
| --- | --- | --- |
| **Valoarea în zecimal** | **Valoarea în hexazecimal** | **Numărul binar coresp. cifrei hexa** |
| 0 | 0 | 0000 |
| 1 | 1 | 0001 |
| 2 | 2 | 0010 |
| 3 | 3 | 0011 |
| 4 | 4 | 0100 |
| 5 | 5 | 0101 |
| 6 | 6 | 0110 |
| 7 | 7 | 0111 |
| 8 | 8 | 1000 |
| 9 | 9 | 1001 |
| 10 | A | 1010 |
| 11 | B | 1011 |
| 12 | C | 1100 |
| 13 | D | 1101 |
| 14 | E | 1110 |
| 15 | F | 1111 |

Mai trebuie ţinut cont la trecerea unui număr prin bazele 2 şi 16 că gruparea cifrelor din baza 2 se face de la dreapta la stânga (prin completare cu zerouri la stânga numărului dacă este cazul, deci în partea care nu-i afectează valoarea.

***Exemple:***

1. Să se convertească 347 din baza 10 în bazele 2 şi 16.

347(D) = 15B(H) = 0001 0101 1011(B)

***Alte exemple:***

2 = 2(10) = 10(2)

62(10) = 111110(2)

1995(10) = 11111001011(2)

1024(10) = 10000000000(2)

***Conversia unui număr dintr-o bază oarecare în baza 10***

Pentru a converti un număr dintr-o bază oarecare în baza 10 se poate folosi formula definită în prima parte a lucrării şi anume:

**Nr(b) = Cn Cn-1 Cn-2 … C2C1 C0**

atunci valoarea sa în baza 10 va fi:

**Nr(10) = Cn \* bn + Cn-1 \* bn-1 + … + C2 \* b2 +  C1 \* b1+ C 0**

***Exemple:***

1. Se dă numărul întreg în hexazecimal 3A8(H) şi se cere valoarea sa în zecimal:

N = 3\*162 + 10\*16 1 + 8 = 3\*256 + 160 + 8 = 936(10)

1. Se dă numărul întreg în hexazecimal 86C(H) şi se cere valoarea sa în zecimal:

86C(16) = 8 \* 162 + 6 \* 16 + 12 = 2156(10)

1. Se dă numărul întreg în binar 1101101(2) și se cere valoarea sa în zecimal

1101101(2) = 1\*26+1\*25+0\*24+1\*23+1\*22+0\*2+1=109(10)

***Alte exemple:***

1010011(2) = 83(10)

11100011(2) = 227(10)

1000000000(2) = 512(10)

11001(2) =25(10)

**Bitul. Bitul de semn. Codul complementar. Regula de reprezentare a numerelor întregi cu semn**

***Bitul***

* Prin particularităţile ei, aritmetica binară se pretează la automatizare mai bine decât aritmetica în orice altă bază de numeraţie. Acesta este motivul pentru care în calculatoare se foloseşte aritmetica în baza 2. În continuare vom folosi alternativ termenul de bit ca sinonim pentru cifră binară.
* **Bitul = unitatea elementară de informație**
* Într-un bit se poate reprezenta o informație care poate să aibă doar două valori posibile: 0 şi 1.
* În funcție de context, un bit poate să însemne 0 sau 1, true sau false, bine sau rău, etc. Totul depinde de interpretarea care se dă bitului!
* Un **octet/byte** este o succesiune de 8 biți, numerotați de la 0 la 7 ca în figura:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 bitul high | 6 | 5 | 4 | 3 | 2 | 1 | 0 bitul low |

***Bitul de semn. Codul complementar***

Dacă interpretăm o anumită configuraţie de biţi drept întreg cu semn, atunci prin convenţie, pentru reprezentarea semnului unui număr se foloseşte un singur bit. Este vorba de bitul high (bitul 7) din octetul high al locaţiei în care se reprezintă numărul. Dacă valoarea acestui bit este zero atunci numărul este pozitiv. Dacă bitul este unu, atunci numărul este negativ.

**Cum se reprezintă numerele întregi în convenţia cu semn?**

S-au propus diferite direcții pe parcursul timpului:

* **Cod direct**: reprezentarea valorii absolute a numărului pe n-1 biţi din cei n ai locaţiei, iar în bitul cel mai semnificativ să se pună semnul Deși această soluție e foarte apropiată de cea naturală, s-a dovedit a fi mai puțin eficientă decât altele. O problemă ar fi faptul că în această reprezentare: -7 + 7 ≠ 0
* **Cod invers (complement față de 1)**: reprezentarea valorii absolute a numărului pe n-1 biţi din cei n ai locaţiei, iar în cazul în care numărul este negativ, să se inverseze toţi cei n biţi ai reprezentării. şi la această reprezentare s-a renunțat, fiind ineficientă (de ex. apare din nou problema -7 + 7 ≠ 0)
* **Cod complementar (complement față de 2)**: Pentru complementarea unui număr întreg reprezentat pe n biţi, mai întâi se inversează valorile tuturor biţilor (valoarea 0 devine 1 şi valoarea 1 devine 0) din locaţia de reprezentare, după care se adaugă 1 la valoarea obţinută.

A treia direcție este şi cea care s-a impus pentru reprezentarea numerelor întregi cu semn.

**Reguli alternative de complementare:**

* Se lasă neschimbaţi biţii începând din dreapta reprezentării binare până la primul bit 1 inclusiv; restul biţilor se inversează până la bitul n-1 inclusiv.
* sau
* Se scade binar conţinutul (evident binar) al locaţiei de complementat din 100 ...00, unde numărul de după cifra binară 1 are atâtea zerouri câţi biţi are locaţia de complementat.
* sau
* Se scade hexazecimal conţinutul (evident hexazecimal) al locaţiei de complementat din 100...00, unde după cifra hexazecimală 1 apar atâtea zerouri câte cifre hexazecimale are locaţia de complementat.

Cele 3 reguli alternative de complementare sunt echivalente cu regula de complementare, dată mai sus prin definiție.

De exemplu, dacă vrem să complementăm o locaţie de un octet şi care conţine numărul (18)10:

|  |  |
| --- | --- |
| Locaţia iniţială:  După inversarea biţilor  Se adaugă  1  Complementul: | 00010010 11101101 11101101+ 00000001 11101110 |

Deci numărul (18)10, adică (12)16, adică (00010010)2, are ca şi complement numărul (11101110)2 adică (EE)16, adică (238)10.

Aplicând regula de scădere binară, avem:

|  |  |
| --- | --- |
| Locaţia iniţială: Complementul: | 100000000-  00010010  11101110 |

Aplicând regula de scădere hexazecimală, avem:

|  |  |
| --- | --- |
| Locaţia iniţială: Complementul: | 100-  12   EE |

***Regula de reprezentare a numerelor întregi cu semn***

Un număr întreg între **-2n-1** şi **2n-1-1** se reprezintă într-o locaţie de **n** biţi astfel:

* dacă numărul este pozitiv, atunci în locaţie se reprezintă numărul respectiv scris în baza 2;
* dacă numărul este negativ, atunci în locaţie se înscrie complementul reprezentării în baza 2 a numărului.

Înainte de a trece la exemple, trebuie să clarificăm situaţia reprezentării numărului  **-2n-1** . Valoarea lui absolută nu poate fi reprezentată pe **n-1** biţi ca să rămână loc şi pentru bitul de semn, ci el se reprezintă pe **n** biţi şi este **100...0.**

Pe de o parte această reprezentare indică un număr negativ! Pe de altă parte, am arătat deja că acest număr este propriul lui complement. Din aceste motive, prin convenţie s-a stabilit că numărul  **-2n-1** se reprezintă în cod complementar pe n biţi prin  **100...0.**  Aceeaşi configuraţie interpretată fără semn reprezintă numărul  **2n-1.**

Tabelul următor prezintă reprezentările mai multor numere, în locaţii de 8 biţi – 1 octet, 16 biţi – 2 octeţi şi 32 de biţi – 4 octeţi.

|  |  |  |  |
| --- | --- | --- | --- |
| Dim. locaţie (octeţi) | Număr în baza 10 | Reprezentare în cod complementar (hexazecimal) | Reprezentare în cod complementar (binar) |
| 1 | 0 | 00 | 00000000 |
| 2 | 0 | 0000 | 0000000000000000 |
| 1 | 1 | 01 | 00000001 |
| 2 | 1 | 0001 | 0000000000000001 |
| 1 | -1 | FF | 11111111 |
| 2 | -1 | FFFF | 1111111111111111 |
| 1 | 127 | 7F | 01111111 |
| 2 | 127 | 007F | 0000000001111111 |
| 1 | -128 | 80 | 10000000 |
| 2 | -128 | FF80 | 1111111110000000 |
| 2 | 128 | 0080 | 0000000010000000 |
| 2 | 32767 | 7FFF | 0111111111111111 |

***De ce codul complementar?***

Implementările operaţiilor peste întregi trebuie, pe de o parte să fie eficiente, iar pe de altă parte să se folosească, pe cât posibil, algoritmi comuni de evaluare a operaţiilor fundamentale peste numere întregi, indiferent de convenţia de reprezentare.

Până în prezent reprezentarea în cod complementar răspunde cel mai bine cerinţelor de mai sus. Principalele motive sunt următoarele două:

* Operaţia de adunare se execută la fel, indiferent de faptul că avem de-a face cu convenţia de reprezentare fără semn sau cea de reprezentare cu semn. Operaţia executată este o adunare simplă, pe n biţi (n – dimensiunea locaţiei), cu ignorarea ultimului transport.
* Operaţia de scădere se reduce la operaţia de adunare a descăzutului cu complementul scăzătorului.

Operațiile de înmulțire şi împărțire se efectuează cu algoritmi separați pentru reprezentările cu şi fără semn, dar procentul operațiilor de adunare şi scădere este mult mai mare în aplicații decât cel al operațiilor multiplicative. De aici şi preferința proiectanților pentru adoptarea codului complementar pentru reprezentarea întregilor cu semn. Un alt avantaj este faptul ca dacă se lucrează cu numere pozitive (și se știe asta de la început), nu se pierde un bit pentru semn. Intervalul de reprezentare este mult mai mare.

**Dimensiunea reprezentării**

Fiind vorba de calcule efectuate cu o maşină, se impun firesc o serie de restricţii legate de reprezentarea întregilor (şi nu numai).

Cea mai importantă dintre ele este dimensiune a reprezentării, adică numărul maxim de cifre binare (numărul de biţi) din reprezentarea unui număr întreg. Să notăm cu ***n*** această dimensiune de reprezentare.

Valorile lui ***n*** la calculatoarele actuale pot fi: 8, 16, 32 şi 64.

Problemă: dacă avem un număr reprezentat pe 7 biți, cum îl reprezentăm pe 8 biți?

Răspuns: depinde de interpretare

* În reprezentarea fără semn completăm cu zerouri biții high rămași
* În reprezentarea cu semn completăm cu bitul de semn biții high rămași

Exemplu: (10011)2 se reprezintă pe un octet ca și:

* + - (00010011)2 în reprezentarea fără semn
    - (11110011)2 în reprezentarea cu semn

Tabelul următor prezintă intervalele de numere reprezentabile într-o locaţie, atât în convenţia fără semn, cât şi în convenţia cu semn, în funcţie de dimensiunea acesteia.

|  |  |  |
| --- | --- | --- |
| Nr. Octeţi | Convenţia fără semn | Convenţia cu semn |
| 1 | [0, 28-1]  = [ 0 , 255 ] | [-27 , 27-1]    = [-128 , 127] |
| 2 | [0, 216-1] = [ 0 , 65535 ] | [-215 , 215-1] = [-32 768 , 32 767] |
| 4 | [0, 232-1] = [ 0 , 4 294 967 295 ] | [-231 , 231-1] = [-2 147 483 648 , 2 147 483 647] |
| 8 | [0, 264-1] = [ 0 , 18 446 824 753 389 551 615 ] | [-263 , 263-1] = [-9 223 412 376 694 775 808 , 9 223 412 376 694 775 807] |

**Instrumente de lucru pentru laborator**

***Instrumentele de lucru pentru laborator sunt următoarele:***

* Editor: Notepad++
* Asamblor: NASM
* Linker: ALINK
* Debugger: Olly DBG

Setul de instrumente poate fi descărcat de aici: [🔗 ASM tools](http://www.cs.ubbcluj.ro/~vancea/asc/asc.2018-2019/files/ASM_tools.zip). Studenții care folosesc alte sisteme de operare decât Windows pot utiliza fie un emulator (tool-urile au fost testate cu succes cu Wine), fie pot rula tool-urile într-o mașină virtuală de Windows. Pentru a ușura redactarea programelor ASM, setul de instrumente include un editor vizual (Notepad++) care are integrat un plugin ce permite efectuarea majorității operațiilor din câteva combinații de taste.

***Instrucțiuni de utilizare:***

* Descărcați setul de instrumente şi extrageți conținutul arhivei
* Porniți editorul Notepad++ din directorul *npp* (nu utilizați versiunea proprie a editorului)
* Procedați la redactarea programelor ce rezolva problemele care va vor fi comunicate. Puteți utiliza următoarele combinații de taste ce se regăsesc în meniul Plugins -> ASM Plugin:

|  |  |  |
| --- | --- | --- |
| ASM Code Template | Ctrl+Shift+N | Completează în editor un program minimal în ASM |
| Build ASM | Ctrl+F7 | Asamblează programul curent |
| Run program | Ctrl+F6 | Rulează programul curent |
| Debug program | F6 | Depanează programul curent, folosind Olly Debugger (documentația pentru debugger se regăsește în directorul *ollydbg* din arhivă |

**Un exemplu minimal de program în limbaj de asamblare**

### *Exemplu:*

bits 32 ;asamblare şi compilare pentru arhitectura de 32 biți

; definim punctul de intrare in programul principal

global start

; declaram funcțiile externe necesare programului nostru

extern exit ; indicam asamblorului ca exit exista, chiar daca noi nu o vom defini

import exit msvcrt.dll ; exit este o funcție care încheie procesul, este definita in msvcrt.dll

; msvcrt.dll conține exit, printf şi toate celelalte funcții C-runtime importante

; segmentul de date in care se vor defini variabilele

segment data use32 class=data

; ...

; segmentul de cod

segment code use32 class=code

start:

; ...

; exit(0)

push dword 0 ; se pune pe stiva parametrul funcției exit

call [exit] ; apelul funcției exit pentru terminarea execuției programului

# Laborator 1 - Probleme propuse

## Rezolvați pe hârtie și explicați următoarele exerciții:

1. Convertiți din baza 10 în 2 și apoi în 16 următoarele numere:

* 4
* 10
* 15
* 32

2. Convertiți din baza 10 în 16 și apoi în 2 următoarele numere:

* 3
* 11
* 16
* 17

3. Convertiți din baza 2 în baza 16 următoarele numere:

* 1010
* 0111
* 1111
* 10001010
* 110101111

4. Convertiți din baza 16 în baza 2 următoarele numere:

* 3
* A
* F
* 2B
* 2F8

5. Efectuați următoarele operatii în baza 2 (fără a converti în baza 10):

* 1+1
* 10+10
* 111+1
* 1010-1
* 1000-10

6. Efectuați următoarele operatii în baza 16 (fără a converti în baza 10):

* 9+1
* B+2
* F+1
* 10+A
* 10-2
* B-3

7. Verificați, folosind cel putin doua dintre regulile de complementaritate, dacă:

* într-o locaţie de 2 octeţi numerele (9A7D)16 și (7583)16 sunt complementare.
* într-o locație de 4 oteți numerele (000F095D)16 și (FFF0F6A3)16 sunt complementare
* Intr-o locație de 2 octeți numerele (4BA1)16  și (5C93)16  sunt complementare
* într-o locație de 1 octet numerele (7F)16 și (81)16 sunt complementare
* Intr-o locatie de 2 octeți numerele (732A)16  și (4E58)16  sunt complementare

8. Reprezentați următoarele numere pe un octet, in reprezentarea cu semn și reprezentarea fără semn:

(11001)2  
(1010)2  
(11001)2  
(1011)2  
(1110111)2  
(10111)2  
(110)2  
(11)2

9. Reprezentați următoarele numere pe doi octeti, in reprezentarea cu semn și reprezentarea fără semn:

(11001010)2  
(1110010)2  
(11001010)2  
(1010110)2  
(11010)2  
(10010)2  
(0010111)2  
(010101)2