
Introducere în organizarea calculatoarelor și limbaje de asamblare (IOCLA)

Reprezentarea datelor în sistemele de calcul

Modificat: Oct-9-18

Cuprins

- Date numerice (întregi)
- Reprezentarea numerelor în baza 2
- Operații cu numere în baza 2
- Reprezentarea numerelor în baza 16
- Reprezentarea numerelor cu semn
- Reprezentarea tipurilor de date de nivel înalt

Suport

- Introduction to Assembly Language Programming
 - * Anexa A, fără 4.1-A4.3

DATE NUMERICE (ÎNTREGI)

Ce reprezentăm cu “date”?

- instrucțiuni - prin coduri de instrucțiuni
- date
 - * logice: Adevarat/Fals, Închis/Deschis, Pornit/Oprit
 - * numerice: întregi, fracționare, pozitive/negative
 - * alfanumerice: caractere, text
 - * multimedia: sunet, imagine (audio/video)
- structuri de date

Date numerice

- Tipurile de bază
 - * Caractere, întregi, **unele** numere fracționare (floating point)
 - * Toate celelalte tipuri existente sunt derivate din tipurile de bază
- Tipurile întregi
 - * Numere cu semn sau fără semn
 - » Cele cu semn sunt considerate pozitive
 - * Au reprezentare pe un număr de biți (8, 16, 32, 64 de biți)
- Tipurile fracționare
 - * Virgulă fixă sau virgulă mobilă (floating point)
 - * Uzual reprezentare în virgulă mobilă
 - * Vor fi detaliate în capitolul 9

Lungime tipuri de date întregi

- signed/unsigned char
 - * lungime 1 octet (8 biți) (arh. pe 32 și 64 de biți)
- signed/unsigned short int
 - * 2 octeți (16 biți) (arh. pe 32 și 64 de biți)
- signed/unsigned int
 - * lungime 4 octeți (32 biți) (arh. pe 32 și 64 de biți)
- signed/unsigned long int
 - * lungime 4 octeți (32 biți) (arh. pe 32 de biți)
 - * Lungime 8 octeți 64 de biți) (arh. pe 64 de biți)
- signed/unsigned long long int
 - * lungime 8 octeți (64 biți) (arh. pe 32 și 64 de biți)

Numere întregi fără semn

- Reprezentare prin simboluri
 - * Modelul de scriere romană a numerelor
 - IL = 59
 - XXXXXIX = 59
 - * Dezavantaj: nu neapărat o reprezentare unică, spațiu mare ocupat
- Reprezentare pozițională
 - * Reprezentare unică
 - * Alegerea unei baze de numeratie
 - » În funcție de criterii subiective sau obiective
 - » Ex: în prezent baza **10**, fenicienii aveau baza **60**, calculatoarele utilizează baza **2**

Baze de numerație

- Reprezentarea pozitionala utilizeaza baze de numeratie si cifre care au valoare diferita in functie de pozitie
 - Ex: fie baza de numeratie **B**
 - **$B > 1$** si **$0 \leq x_{n-1}, x_{n-2}, x_{n-3}, \dots, x_1, x_0 < B$**
- avem reprezentarea unica in baza B : $x_{n-1}x_{n-2}x_{n-3} \dots x_1x_0 (B)$
- unde: $x_{n-1}x_{n-2}x_{n-3} \dots x_1x_0 (B) = x_{n-1} * B^{n-1} + x_{n-2} * B^{n-2} +$
 - $\dots + x_1 * B^1 + x_0 * B^0$

Reprezentarea numerelor în baza 10

- $2891 = 2 \cdot 10^3 + 8 \cdot 10^2 + 9 \cdot 10^1 + 1$
- Preferată pentru specificul uman (numărat pe degete)

REPREZENTAREA NUMERELOR ÎN BAZA 2

Reprezentarea numerelor în baza 2

- Reprezentare binară
- Ușor de implementat pe calculator
- Două niveluri: închis/deschis, sus/jos
 - * circuite digitale (vezi curs de electronică)
 - * bit = 0 -> bitul **nu este** sub tensiune
 - * bit = 1 -> **este** sub tensiune
- Bit = Binary Digit

Formate binare de reprezentare

- Bit
 - * binary digit
 - * unitatea elementară de informație
 - * starea unui bistabil, sau a unei celule elementare de memorie
- Octet (byte)
 - * grup de 8 biți
 - * unitatea elementara de adresare la cele mai multe calculatoare actuale (inclusiv Intel x86)
 - * poate reprezenta o valoare numerică, un caracter (ASCII)

Conversii din baza 2 în baza 10

- 10111010: $1*2^7 + 0*2^6 + 1*2^5 + 1*2^4 + 1*2^3 + 0*2^2 + 1*2^1 + 0*2^0 = 186$
- Rapide:
 - * 10000000:?
 - * 00100000:?
 - * 00011111:?
 - * 00001111:?
 - * 00001110:?

Conversii din baza 10 in baza 2

- Clasic: se împarte la doi, se reține bitul de rest; apoi se ia câtul, se împarte la 2, etc.
- Mai ingineresc: se adună succesiv puteri ale lui 2
 - * $200 = 128 + 64 + 8 = 11001000$
 - * $163 = 128 + 32 + 2 + 1 = 10100011$
- Rapid:
 - * 32: ?
 - * 65: ?
 - * 15: ?
 - * 14: ?

OPERAȚII CU NUMERE ÎN BAZA 2

Operații aritmetice în baza 2

- Adunare, scădere, împărțire, înmulțire
- Aceleași principii ca la baza 10
- Făcute de calculator
- Improbabil să fie nevoie de făcut de noi
 - * Dacă este cazul, facem conversia înainte și înapoi în/din baza 10

Operatii pe biți

- Bitwise operations
- Bitwise OR vs. Logical OR
- AND
- NOT
- XOR (Exclusive OR)

Deplasări (shift)

- Shift right (SHR)
- Shift left (SHL)
- shift logic vs. shift aritmetic
 - * clarificăm la numere cu semn
- Ce înseamnă SHR N, 2 (shift la dreapta cu 2)?
- Ce înseamnă SHL N, 2 (shift la dreapta cu 2)?

Măști de biți

- Valori predefinite
- Folosite pentru verificare
- Fie masca 11110000
 - * Ce înseamnă $N \& \text{masca}$? (bitwise AND)
 - * Ce înseamnă $N | \text{masca}$? (bitwise OR)
- Ce înseamnă $N | 0$, $N \& 0$, $N \& \text{MAX}$, $N | \text{MAX}$?
- Ce înseamnă $N \wedge N$? (eXclusive OR)
- Ce înseamnă $N \wedge 0$, $N \wedge \text{MAX}$?

Exemple practice de operatii pe biți

- Rețelistică (mască de subrețea, adresa de subrețea)
- Hărți de biți (bitmaps): pentru spațiu ocupat puțin: gestiunea spațiului liber pe disc
- Opțiuni active sau nu: permisiuni Unix pe fișiere

REPREZENTAREA NUMERELOR ÎN BAZA 16

Reprezentarea numerelor în baza 16

- Numită reprezentare în hexazecimal
 - * Informal: “în hexa”
- De ce este utilă?
 - * Formă compactă a reprezentării binare
 - * 4 biți formează o cifră hexa (nibble)
- 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- Reprezentare în ingineria calculatoarelor
 - * `0x89AB`
 - * `89ABh`

Scenarii de utilizare reprezentare hexa

- Dump hexa pentru date binare
 - * Utilitare precum xxd, hexdump, od
- Vizualizare adrese de memorie
 - * Dezasamblare
 - * Debugging
 - * Hardware
- Generare date binare
 - * Demo echo, python, perl (pe Linux)

Conversii baza 2 <-> baza 16

- 2 -> 16
 - * Se iau câte 4 biți și se transformă în nibble
 - * 1101: B, 1110: E, 11000101: C5
- 16 -> 2
 - * Se transformă fiecare nibble în șir de 4 biți
 - * A: 1010, C: 1100, 87: 10000111
- Exerciții rapide
 - * 11000000 10101000: ?
 - * 10010011 00111110: ?
 - * ABCD: ?
 - * 9876: ?

Conversii baza 10 <-> baza 16

- De obicei are sens pentru numere până în 256
- 10 -> 16
 - * Se împarte la 16 și se folosesc câtul și restul
 - * $190 \rightarrow 16 * 11 + 14 \rightarrow 16 * B + E \rightarrow 0xBE$
 - * $95 \rightarrow 16 * 5 + 15 \rightarrow 16 * 5 + F \rightarrow 0x5F$
- 16-> 10
 - * Înmulțire cu 16 a primei cifre și adunarea ultimei
 - * $AA \rightarrow 16 * A + A \rightarrow 160 + 10 \rightarrow 170$
 - * $CC \rightarrow 16 * C + C \rightarrow 192 + 12 \rightarrow 204$

- Demo în gdb

- * `set $i = 26`
- * `p/d $i*2`
- * `p/x $i`
- * `p/t $i`
- * `p/d 0x22`
- * `help x`
- * `set $j = -11`
- * `p/d ($j & 0xff)`
- * În sasm: add pe byte 11 + 245

Împachetarea datelor (endianess)

- LSB: Least Significant Byte
- MSB: Most Significant Byte
- Little endian: LSB la adresa de memorie cea mai mică
- Big endian : MSB la adresa de memorie cea mai mică
- Important la transferul de date între sisteme diferite; de ales un format comun

- x86: little endian
- Internet: big endian

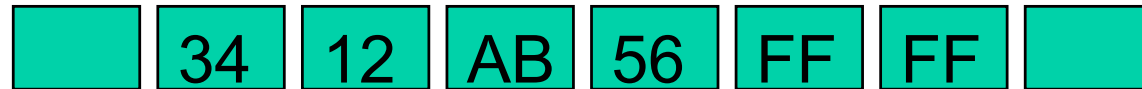
Reprezentare Little Endian

* Octeti: 3AH, 33H, 12H



Adresă: x x+1 x+2 x+3 x+4

* Cuvinte: 1234H, 56ABH, FFFFH



Adresă: x x+1 x+2 x+3 x+4 x+5 x+6 x+7

* Dublu-cuvinte: 01234567H, 89ABCDEFH



Adresă: x x+1 x+2 x+3 x+4 x+5 x+6 x+7 x+8 x+9

- Demo în SASM

- * `v1 db 1, 2, 3, 4, 0xa, 0xb, 0xc, 0xd`
- * `v2 dw 0x0102, 0x0304, 0x0a0b, 0x0cod`
- * `v3 dd 0x01020304, 0xabcd6789`
- * Examinare cu `x/20xb &v1`, `x/20xh &v1`, `x/20xw &v1`, etc

REPREZENTAREA NUMERELOR CU SEMN

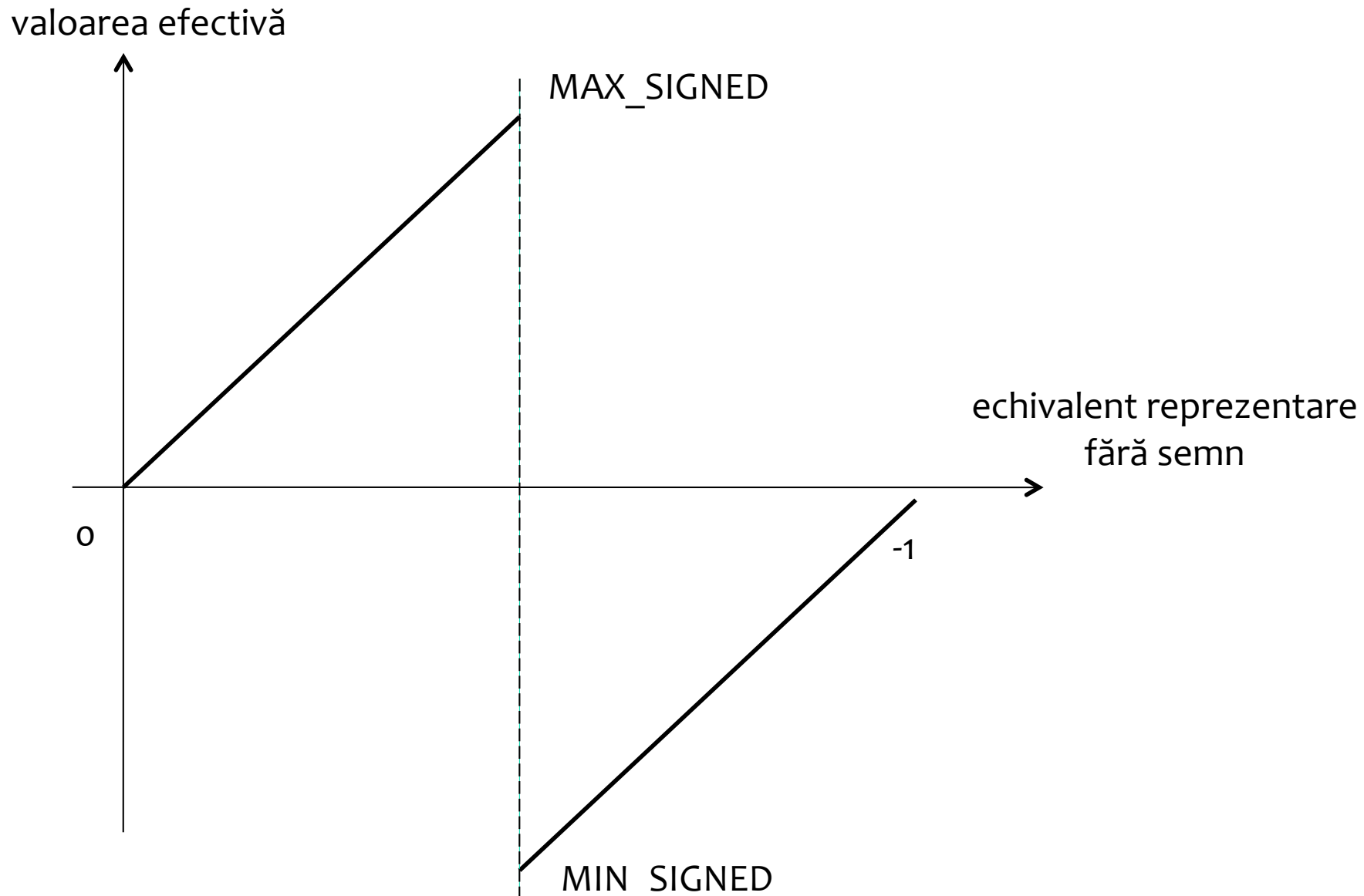
Cât înseamnă 0xFF?

- Depinde
 - * $F*16+F = 255$, pentru număr fără semn
 - * -1 pentru reprezentare cu semn
- Într-o reprezentare cu semn toți biții sunt parte a numărului (fiecare bit are o pondere valorică)

Ideea de reprezentare a numerelor cu semn

- Se alocă primul bit pentru semn
 - * 0: număr pozitiv
 - * -1: număr negativ
- Restul biților dau valoarea

Intuiția reprezentării numerelor cu semn



Exemple de reprezentare cu semn

- 8 biți

- * 00000000 (0x00): 0
- * 01111111 (0x7F): 127 (SCHAR_MAX)
- * 10000000 (0x80): -128 (SCHAR_MIN)
- * 11111111 (0xFF): -1

- 16 biți

- * 00000000 00000000 (0x0000): 0
- * 01111111 11111111 (0x7FFF): 32767 (SHRT_MAX)
- * 10000000 00000000 (0x8000): -32768 (SHRT_MIN)
- * 11111111 11111111 (0xFFFF): -1

Complement față de 2

- Sistemul uzual de reprezentare a numerelor cu semn
- Îl vom numi C2
- Numerele pozitive rămân neschimbate
- Numerele negative
 - * Au ca prim bit 1 (bitul de semn)
 - * Algoritm
 - » 1. Se neagă tot șirul de biți reprezentând numărul în modul
 - » 2. Se adună 1 la valoarea respectivă, ca la adunările între numere întregi fără semn
 - » 3. Se trunchiază șirul de biți la dimensiunea inițială

Caracteristici C2

- Pentru o reprezentare pe N biți, acoperă plaja
$$[-2^{N-1}, 2^{N-1} - 1]$$
 - * Pentru 8 biți: [-128, 127]
 - * Pentru 16 biți: [-32768, 32767]
- Dacă se incrementează fără semn MAX (reprezentarea maximă) se ajunge în MIN (reprezentarea minimă)
 - * Demo
- Reprezentarea lui -1 e echivalentul celui mai mare număr dacă ar fi reprezentarea fără semn (numai biți de 1)

Exemple de reprezentare de numere în C2

- Pe 8 biți
- -20 (not 20 + 1)
 - * 20: 00010100
 - * not 20: 11101011
 - * -20 în C2 (not 20 + 1): 11101011 + 1: 11101100
- Ce reprezintă 11010101 (0xD5), dacă știm că numărul e cu semn?
 - * Primul bit: 1, negativ
 - * R-1: 11010101 - 1 = 11010100
 - * -N = not (R-1) = 00101011 = 43
 - * N = -43

Incrementări interesante în C2

- $\text{SCHAR_MAX} + 1 = \text{SCHAR_MIN}$
 - * $01111111 + 1 = 10000000$
 - * $\text{signed_value}(10000000) = -(\text{NOT } 10000000 + 1) = -(01111111 + 1) = -10000000 = -0x80 = -128 (\text{SCHAR_MIN})$
- $-1 + 1 = 0$ (Duh!)
 - * $11111111 + 1 = 00000000$

Operații aritmetice în C2

- `SCHAR_MIN + SCHAR_MAX`
 - * $10000000 + 01111111 = 11111111 (-1)$
- $-1 + 10$
 - * $11111111 + 00001010 = 00001001$
- $100 + 100$ (ambele numere cu semn)
 - * $01100100 + 01100100 = 11010000 \rightarrow -56$
 - * se schimbă bitul de semn de la doi operanzi de semn opus
 - » se activează “overflow flag”
 - » informație că rezultatul este “incorect”

Ce înseamnă?

- `0xFE`
- `(unsigned char) -1`
- `(int) 0xff`
- `char c = 0xff; int d = c;`

Curiozate: INT_MIN % -1

- Demo în SASM:
 - * add, sub pe byte
 - * activare CF, OF

REPREZENTAREA TIPURILOR DE DATE DE NIVEL ÎNALT

Structuri

- Date contigue
- Date de diferite tipuri (dimensiuni, cu semn/fără semn)
- O dată este la un anumit deplasament (offset) față de adresa de început a structurii

Vectori (arrays)

- O înșiruire (de obicei compactă) de date
- Vector de întregi (32 de biți)
 - * La fiecare 4 octeți avem o nouă valoare întreagă
 - * $A, A+1, A+2, A+3$ (little sau big endian)
- `sizeof(a)`: dimensiunea ocupată de toate elementele vectorului

Șiruri

- Vector de caractere
- Ultimul caracter este NUL-terminatorul ('\0', valoarea 0x0)
- `char s[] = "ana"; /* 4 bytes; NUL-terminated */`
- `char s[] = {'a', 'n', 'a'}; /* 3 bytes */`
- `char s[] = "\xb0\xcd\xdc"; /* string (non ASCII) */`

Cuvinte cheie

- Date întregi
- Baze de numerație
- Baza 10
- Baza 2
- Baza 16
- Conversii
- Numere cu semn
- Numere fără semn
- Endianness
- Bit, octet
- Operații logice (pe biți)
- Mască de biți
- Deplasare (shift)
- Nibble
- Complement față de 2
- Structuri, vectori, șiruri