**Curs 1 și 2**

Contents

[1. Sisteme de numerație și coduri 3](#_Toc18488270)

[1. Nummerierungs- und Kodierungssysteme 3](#_Toc18488271)

[1.1 Sisteme de numerație 3](#_Toc18488272)

[1.1 Nummerierungssysteme 3](#_Toc18488273)

[1.2 Conversia bazei de numerație 3](#_Toc18488274)

[1.2 Umwandlung der Zahlenbasis 3](#_Toc18488275)

[1.3 Coduri binare 6](#_Toc18488276)

[1.3 Binäre Codes 6](#_Toc18488277)

[1.3.1 Coduri ponderate 6](#_Toc18488278)

[1.3.1 Gewichtete Codes 6](#_Toc18488279)

[1.3.2 Coduri neponderate 7](#_Toc18488280)

[1.3.2 Nicht gewichtete Codes 7](#_Toc18488281)

[1.3.3 Bitul de paritate pentru detecția erorilor 8](#_Toc18488282)

[1.3.3 Paritätsbit für Erkennen von Fehlern 8](#_Toc18488283)

[2. Reprezentarea numerelor în calculator 9](#_Toc18488284)

[2. Darstellung der Zahlen in den Computer 9](#_Toc18488285)

[2.1 Întregi fără semn (*Unsigned Integers*) 10](#_Toc18488286)

[2.1 Vorzeichenlose ganze Zahlen (*Unsigned Integers*) 10](#_Toc18488287)

[2.2 Complementul lui 2 (*Two’s Complement*) 11](#_Toc18488288)

[2.2 2-Komplement Ganzzahl (*Two’s Complement*) 11](#_Toc18488289)

[2.3 Fracționare fără semn (*Unsigned fractional*) 14](#_Toc18488290)

[2.3 Vorzeichenlose Bruchzahlen (*Unsigned fractional*) 14](#_Toc18488291)

[2.4 Fracționare în complementul lui 2 (*Two’s Complement signed fractional*) 17](#_Toc18488292)

[2.4 2-Komplement vorzeichen Bruchzahlen (*Two’s Complement signed fractional*) 17](#_Toc18488293)

[2.5 Codul Gray 17](#_Toc18488294)

[2.5 Gray Code 17](#_Toc18488295)

[2.6 Mărime și semn (*Signed magnitude*) 19](#_Toc18488296)

[2.6 Vorzeichenbehaftete Größenordnung (*Signed magnitude*) 19](#_Toc18488297)

[2.7 Complementul lui 2 cu deplasament (*Offset two’s Complement*) 20](#_Toc18488298)

[2.7 Offset Zweierkomplement (*Offset two’s Complement*) 20](#_Toc18488299)

[2.8 Complementul lui 1 (*One’s Complement*) 20](#_Toc18488300)

[2.8 Einerkomplement (*One’s Complement*) 20](#_Toc18488301)

[2.9 Virgulă mobilă (*Floating point*) 21](#_Toc18488302)

[2.9 Gleitkomma-Zahlen (*Floating point*) 21](#_Toc18488303)

[3. Reprezentări binare și ordini de plasare 25](#_Toc18488304)

[3. Binäre Darstellungen und Platzierungsaufträge 25](#_Toc18488305)

[3.1 Dimensiunea reprezentării 25](#_Toc18488306)

[3.1 Darstellungsdimension 25](#_Toc18488307)

[3.2 Organizarea și memorarea datelor 26](#_Toc18488308)

[3.2 Daten organisieren und speichern 26](#_Toc18488309)

[3.3 Tipuri elementare de date: dimensiuni ale standardelor de reprezentare 31](#_Toc18488310)

[3.3 Elementare Datentypen: Dimensionen von die Darstellungsstandarden 31](#_Toc18488311)

[3.4 Ordinea octeților într-o locație; mașini little-endian și mașini big-endian 32](#_Toc18488312)

[3.4 Die Reihenfolge der Bytes an einem Speicherort; little-endian und big-endian Maschinen 32](#_Toc18488313)

[3.5 Unități de capacitate a memoriei 37](#_Toc18488314)

[3.5 Speicherkapazitätseinheiten 37](#_Toc18488315)

[3.6 Codificarea caracterelor 37](#_Toc18488316)

[3.6 Zeichenkodierung 37](#_Toc18488317)

[3.7 Înmulțiri și împărțiri 40](#_Toc18488318)

[3.7 Multiplikationen und Divisionen 40](#_Toc18488319)

[3.8 Conversia la o locaţie de alte dimensiuni 43](#_Toc18488320)

[3.8 An einen anderen Speicherort konvertieren 43](#_Toc18488321)

|  |  |  |
| --- | --- | --- |
| 1. Sisteme de numerație și coduri |  | 1. Nummerierungs- und Kodierungssysteme |
| 1.1 Sisteme de numerație |  | 1.1 Nummerierungssysteme |
| Sistemele numerice prelucrează informația. În vederea prelucrării, informația trebuie să fie *codificată*. Pentru codificare, se utilizează un anumit tip de reprezentare.  *Sistemul de numerație* este format din totalitatea regulilor de reprezentare a numerelor cu ajutorul unor simboluri numite cifre. Sistemele de numerație pot fi *poziționale* (valoarea unei cifre este determinată de poziția sa în cadrul numărului) sau *nepoziționale*.  Un număr „*N*“ într-un sistem pozițional poate fi reprezentat într-o bază de numerație „*b*“ astfel: |  | Numerische Systeme verarbeiten die Informationen. Zur Verarbeitung müssen die Informationen *codiert* werden. Zur Kodierung wird eine bestimmte Art der Darstellung verwendet.  *Das Nummerierungssystem* besteht aus allen Regeln für die Darstellung von Zahlen durch Symbole, die Ziffern genannt werden. Numerische Systeme können *positionell* sein (der Wert einer Ziffer wird durch ihre Position innerhalb der Zahl bestimmt) oder *nicht positionell* sein.  Eine „*N*“ – Nummer in einem Positionssystem kann in einer Zahlenbasis „*b*“ wie folgt dargestellt werden: |
|  | | |
| unde baza „*b*“ este un număr întreg mai mare ca 1 și *ai* sunt întregi în gama . Numărul „*N*“ în baza „*b*” se notează astfel: (*N*)*b*. Atunci când baza nu este specificată, ea este implicit 10 (deoarece sistemul zecimal este cel mai utilizat în practică).  Când baza *b* = 2, reprezentarea numerică se numește *sistem numeric binar*. Complementul unei cifre „a“, notat cu „“, în baza „*b*“ este definit ca . În sistemul numeric binar  și . |  | wobei Basis „b“ eine ganze Zahl größer als 1 ist und ai eine ganze Zahl im Bereich  ist. Die Zahl „N“ in der Basis „b“ lautet wie folgt: (*N*)*b*. Wenn keine Basis angegeben wird, ist der Standard 10 (weil das Dezimalsystem in der Praxis am häufigsten verwendet wird).  Wenn die Basis *b* = 2 ist, wird die numerische Darstellung als binäres numerisches System bezeichnet.  Das Komplement einer Ziffer „a“, gekennzeichnet mit „“, ist in der Basis „b“ als  definiert. Im binären Zahlensystem und . |
| 1.2 Conversia bazei de numerație |  | 1.2 Umwandlung der Zahlenbasis |
| În multe aplicații practice se pune problema conversiei unui număr exprimat în baza „*b*1“ în altă bază „*b*2“. În procesul de conversie distingem două cazuri:  a) *b*1 < *b*2  b) *b*1 > *b*2  În cazul a) conversia implică exprimarea numărului (*N*)*b*1 ca un polinom în puterile lui „*b*1“ și evaluarea polinomului folosind aritmetica în baza „*b*2“. |  | In vielen praktischen Anwendungen besteht das Problem die Umwandlung einer Zahl, die in der Basis "*b*1" ausgedrückt wird, in eine andere Basis "*b*2" zu erfassen. Bei der Umwandlung unterscheiden wir zwei Fälle:  a) *b*1 < *b*2  b) *b*1 > *b*2  Im Falle von 1) die Umwandlung beinhaltet zwei Schritte: der Zahl (*N*)*b*1 als Polynom in Potenzen von "*b*1" und die Auswertung des Polynoms unter Verwendung der auf "*b*2" basierenden Arithmetik zu äußern. |
| **Exemplu**: Pentru *b*1 = 3, *b*2 = 10 și (*N*)3 = 2201.1 vom avea: |  | **Beispiel**: Für *b*1 = 3, *b*2 = 10 und (*N*)3 = 2201.1 haben wir: |
| (*N*)10 = 2⋅33 + 2⋅32 + 0⋅31 + 1⋅30 + 1⋅3-1  = 54 + 18 + 0 + 1 + 0,3 = 73,3 | | |
| În cazul b) este mai convenabil să utilizăm aritmetica în baza „*b*1“. Conversia numărului se face prin conversia separată a părții întregi și a părții fracționare a acestuia. Pentru conversia părții întregi a numărului, aceasta se împarte la baza „*b*2“ obținându-se astfel un cât și un rest; se reține restul și se continuă cu împărțirea la baza „*b*2“ a câtului. Algoritmul se oprește în momentul în care avem câtul 0. Partea întreagă a rezultatului se obține prin scrierea resturilor în ordinea inversă a generării lor.  Pentru conversia părții fracționare a numărului, aceasta se înmulțește cu baza „*b*2“, obținându-se astfel un număr format dintr-o parte întreagă și o parte fracționară; se reține partea întreagă și se continuă cu înmulțirea cu baza „*b*2“ a părții fracționare obținute. Procesul continuă până la obținerea preciziei dorite. |  | Im Fall 2) ist es günstiger, die Arithmetik in der Basis "*b*1" zu verwenden. Die Konvertierung der Nummer erfolgt durch separate Konvertierung des Ganzteils und des Bruchteils. Für die Umwandlung des Ganzteils der Zahl wird diese in die Basis "*b*2" unterteilt, wodurch sowohl ein Quotient als auch ein Rest erhalten wird; der Rest wird beibehalten und mit der Division an der Basis "*b*2" des Quotienten fortgesetzt. Der Algorithmus stoppt, wenn der Quotient 0 ist. Der Ganzteil des Ergebnisses wird durch Schreiben der Reste in umgekehrter Reihenfolge ihrer Erzeugung erhalten.  Um den gebrochenen Teil der Zahl zu konvertieren, multiplizieren Sie ihn mit der Basis "*b*2", um eine Zahl zu erhalten, die aus einem Ganzen und einem gebrochenen Teil besteht. Behalten Sie der Ganzteil und fahren Sie mit der Multiplikation des gebrochenen Teils mit "*b*2" fort. Der Prozess wird fortgesetzt, bis die gewünschte Genauigkeit erreicht ist. |
| **Exemplu**:  Pentru *b*1 = 10, *b*2 = 4 și (*N*)10 = 47.4 vom avea:  1. Conversia părții întregi:  47:4 = 11 rest 3  11:4 = 2 rest 3  2:4 = 0 rest 2  Partea întreagă a rezultatului este (233)4.  2. Conversia părții fracționare:      Partea fracționară a rezultatului este (1212…)4.  În concluzie (*N*)4 = 223.1212… |  | **Beispiel:**  Für *b*1 = 10, *b*2 = 4 und (*N*)10 = 47.4 haben wir:  1. Die Umwandlung des Ganzteils:  47:4 = 11 Rest 3  11:4 = 2 Rest 3  2:4 = 0 Rest 2  Der Ganzteil des Ergebnisses ist: (233)4.  2. Die Umwandlung des Bruchteils:      Der Bruchteil des Ergebnisses ist: (1212…)4.  Abschließend, (*N*)4 = 223.1212… |
| Alte Exemple  1. Pentru *b*1 = 10, *b*2 = 6 și (*N*)10 = 985437 avem:  985437 : 6 = 164239 rest 3  164239 : 6 = 27373 rest 1  27373 : 6 = 4562 rest 1  4562 : 6 = 760 rest 2  760 : 6 = 126 rest 4  126 : 6 = 21 rest 0  21 : 6 = 3 rest 3  3 : 6 = 0 rest 3  Deci avem (985437)10 = (33042113)6.  2. Pentru *b*1 = 10, *b*2 = 16 și (*N*)10 = 985437 avem:  985437 : 16 = 61589 rest 13 (D)  61589 : 16 = 3849 rest 5 3849 : 16 = 240 rest 9  240 : 16 = 15 rest 0  15 : 16 = 0 rest 15 (F)  Deci avem (985437)10 = (F095D)16. |  | Andere Beispiele  1. Für *b*1 = 10, *b*2 = 6 und (*N*)10 = 985437 haben wir:  985437 : 6 = 164239 Rest 3  164239 : 6 = 27373 Rest 1  27373 : 6 = 4562 Rest 1  4562 : 6 = 760 Rest 2  760 : 6 = 126 Rest 4  126 : 6 = 21 Rest 0  21 : 6 = 3 Rest 3  3 : 6 = 0 Rest 3  Abschließend (985437)10 = (33042113)6.  2. Für *b*1 = 10, *b*2 = 16 und (*N*)10 = 985437 haben wir:  985437 : 16 = 61589 Rest 13 (D)  61589 : 16 = 3849 Rest 5 3849 : 16 = 240 Rest 9  240 : 16 = 15 Rest 0  15 : 16 = 0 Rest 15 (F)  Abschließend (985437)10 = (F095D)16. |
| Un caz aparte îl constituie conversia numerelor octale și hexazecimale în binar și invers. Atunci se poate folosi o procedură de conversie mult mai simplă. Pentru aceasta fiecare cifră octală sau hexazecimală se exprimă prin 3, respectiv 4 cifre binare.  **Exemplu**:  (123.4)8 = (001 010 011.100) = (1010011.1) 2  (B7.2) 16 = (1011 0111.0010) = (10110111.001) 2 |  | Ein Sonderfall ist die Umwandlung von Oktal- und Hexadezimalzahlen in Binärzahlen und umgekehrt. Dann kann ein wesentlich einfacherer Umwandlungsverfahren verwendet werden. Zu diesem Zweck wird jede Oktal- oder Hexadezimalziffer durch 3 bzw. 4 Binärziffern ausgedrückt.  **Beispiel:**  (123.4)8 = (001 010 011.100) = (1010011.1)2  (B7.2)16 = (1011 0111.0010) = (10110111.001)2 |
| La conversia din binar în octal sau hexazecimal se fac grupări de câte 3, respectiv 4 cifre binare.  **Exemplu**:  (1010110.0101)2 = (001 010 110.010 100) = (126.24)8  (1011110.011)2 = (0101 1110.0110) = (5E.6)16 |  | Bei der Umwandlung von Binär- in Oktal- und Hexadezimalzahlen werden Gruppen von 3 bzw. 4 Binärziffern gebildet.  **Beispiel:**  (1010110.0101)2 = (001 010 110.010 100) = (126.24)8  (1011110.011)2 = (0101 1110.0110) = (5E.6)16 |
| 1.3 Coduri binare |  | 1.3 Binäre Codes |
| Deși sistemul de numerație binar are multe avantaje practice și este foarte utilizat în calculatoarele numerice, în multe cazuri este convenabil să lucrăm cu sistemul zecimal, în special acolo unde comunicația între om și mașină este intensă. Pentru a simplifica problema comunicației au fost definite un număr de coduri astfel încât cifrele zecimale să fie reprezentate prin succesiuni de cifre binare.  Pentru a reprezenta cele 10 cifre zecimale este suficient să folosim 4 cifre binare. Codurile binare se pot împărți în două clase: *ponderate* și *neponderate*. |  | Obwohl das binäre Nummerierungssystem viele praktische Vorteile hat und in numerischen Computern weit verbreitet ist, ist es oft zweckmäßig, mit dem Dezimalsystem zu arbeiten, insbesondere wenn der Mensch-Maschine-Informationsaustausch intensiv ist. Um das Kommunikationsproblem zu vereinfachen, wurde eine Anzahl von Codes definiert, so dass die Dezimalzahlen durch binäre Ziffernfolgen dargestellt werden. Zur Darstellung der 10 Dezimalstellen verwenden Sie einfach 4 Binärstellen. Binäre Codes können in zwei Klassen unterteilt werden: gewichtet und nicht übereinstimmend. |
| 1.3.1 Coduri ponderate |  | 1.3.1 Gewichtete Codes |
| Caracteristica principală a codurilor ponderate este aceea că fiecărei cifre binare îi este asociată o „pondere“. Pentru fiecare grup de 4 biți suma ponderilor acelor cifre binare a căror valoare este 1 este egală cu cifra zecimală pe care o reprezintă. O cifră zecimală într-un cod ponderat se scrie astfel: |  | Das Hauptmerkmal gewichteter Codes besteht darin, dass jeder Binärzahl eine "Gewichtung" zugeordnet ist. Für jede 4-Bit-Gruppe entspricht die Summe dieser Binärziffern, deren Wert 1 ist, der Dezimalzahl, die sie darstellen. Eine Dezimalzahl in einem gewichteten Code wird wie folgt geschrieben: |
|  | | |
| unde *ai* = 0 sau 1.  În Tabelul 1 se dau trei exemple de coduri binare ponderate. Primul cod se numește **BCD** (**B**inary **C**oded **D**ecimal), deoarece pentru obținerea codului fiecare cifră zecimală este convertită în binar. |  | wobei *ai* = 0 oder 1.  Tabelul 1 enthält drei Beispiele für gewichtete Binärcodes. Der erste Code wird als **B**inär**c**odierte **D**ezimalzahl (**BCD**) bezeichnet, da jede Dezimalstelle in Binärcode umgewandelt wird, um den Code zu erhalten. |
| Tabelul 1. *Trei exemple de coduri ponderate* (Drei Beispiele für gewichtete Codes)   |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | Cifră  Zecimală  (*Dezimalzahl*) | b3 | b2 | b1 | b0 | b3 | b2 | b1 | b0 | pondere negativă  (*negatives Gewicht*) | | | | |  | 8 | 4 | 2 | 1 | 2 | 4 | 2 | 1 | 6 | 4 | 2 | -3 | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | | 3 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | | 4 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | | 5 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | | 6 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | | 7 | 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 1 | | 8 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | | 9 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | | | |
| 1.3.2 Coduri neponderate |  | 1.3.2 Nicht gewichtete Codes |
| Cele mai utilizate coduri neponderate sunt codul *Exces 3* și codul *Gray*. Codul Exces 3 este format prin adăugarea lui 0011 la fiecare cuvânt de cod din BCD. Este un cod autocomplementar și posedă anumite proprietăți care l-au făcut practic. Din acest cod s-a eliminat combinația 0000, care ar putea fi confundată cu lipsa de informație.  În multe aplicații practice, analog conversiei digitale este de dorit a se folosi codurile în care toate cuvintele de cod succesive diferă doar printr-o cifră. Codurile care au o astfel de proprietate se numesc *coduri ciclice*.  Un astfel de cod este codul Gray. Codul Gray de „*n*“ biți face parte din clasa *codurilor reflectate*. Termenul „reflectate“ este folosit pentru a desemna coduri care au următoarea proprietate: cuvântul de cod de „*n*“ biți poate fi generat prin reflectarea codului de „*n*‑1“ biți. |  | Die am meisten verwendete Codes sind Code Excess 3 und Gray Code. Code *Excess 3* wird durch Hinzufügen von 0011 zu jedem BCD-Codewort gebildet. Es ist ein selbstkomplementären Code und besitzt bestimmte Eigenschaften, die ihn praktisch machen. Aus diesem Code ist die 0000-Kombination gelöscht, die mit dem Mangel an Informationen verwechselt werden könnte.  In vielen praktischen Anwendungen, bei der digitalen Umwandlung, ist es wünschenswert, Codes zu verwenden, bei denen sich alle aufeinanderfolgenden Codewörter nur um eine Ziffer unterscheiden. Codes, die über eine solche Eigenschaft verfügen, werden als *zyklische Codes* bezeichnet.  Ein solcher Code ist der Gray-Code. Gray 'n'-Bitcode ist Teil der reflektierten Codeklasse. Der Begriff"*reflektiert*" wird verwendet, um Codes zu bezeichnen, die die folgende Eigenschaft haben: Das Codewort von "*n*" Bits kann durch Reflektion der "*n*-1" Bits Code erzeugt werden. |
| Tabelul 2. *Obținerea codului Gray de 3 biți din cel de 2 biți* (Herstellung des 3-Bit-Gray-Codes von den 2-Bit-Code)   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **Gray 2 biți (*2 Bits*)** | | **Gray 3 biți (*3 Bits*)** | | | | 0 | 0 | 0 | 0 | 0 | | 0 | 1 | 0 | 0 | 1 | | 1 | 1 | 0 | 1 | 1 | | 1 | 0 | 0 | 1 | 0 | |  |  | 1 | 1 | 0 | |  |  | 1 | 1 | 1 | |  |  | 1 | 0 | 1 | |  |  | 1 | 0 | 0 | | | |
| În **Tabelul 3** sunt prezentate cifrele zecimale în cod Exces 3, respectiv Gray: |  | **Tabelul 3** zeigt die Dezimalstellen in Überschuss 3 (Exces 3) oder Gray-Code. |
| **Tabelul 3.** *Codificarea cifrelor zecimale în codurile Exces 3 și Gray* **(**Die Dezimalstellen in Überschuss 3 (Exces 3) oder Gray-Code)   |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **cifră zecimală (*Dezimalzahl*)** | **Exces 3** | | | | **Gray** | | | | | **0** | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | | **1** | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | | **2** | 0 | 1 | 0 | 1 | 0 | 0 | 1 | 1 | | **3** | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | | **4** | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | | **5** | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | | **6** | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | | **7** | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | | **8** | 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | | **9** | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | | | |
| 1.3.3 Bitul de paritate pentru detecția erorilor |  | 1.3.3 Paritätsbit für Erkennen von Fehlern |
| În procesul de transmitere a informației în sistemele numerice, aceasta poate fi alterată. În vederea determinării corectitudinii informației recepționate, se pot folosi coduri detectoare și corectoare de erori.  Codurile detectoare de erori au următoarea proprietate: *apariția unei singure erori transformă un cuvânt valid într-un cuvânt invalid*.  O metodă pentru detecția erorilor este *metoda bitului de paritate*. Ideea de bază în controlul de paritate este de a adăuga o cifră binară în plus la fiecare cod cuvânt al unui cod dat, pentru a face ca numărul de biți de 1 din fiecare cuvânt să fie impar sau par. |  | Bei der Übertragung von Informationen in numerischen Systemen kann diese geändert werden. Um die Genauigkeit der empfangenen Informationen zu bestimmen, können Fehlererkennungs- und Korrekturcodes verwendet werden.  Fehlererkennung Codes haben die folgende Eigenschaft: *Das Auftreten eines einzelnen Fehlers wandelt ein gültiges Wort in ein ungültiges Wort um*.  Eine Methode zum Erkennen von Fehlern ist die *Paritätsbit Methode*. Die Grundidee bei der Paritätskontrolle besteht darin, jedem Wortcode eines gegebenen Codes eine zusätzliche Binärziffer hinzuzufügen, um die Anzahl der Bits von 1 in jedem Wort ungerade oder gerade zu machen. |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | 0 | 0 | 1 | 1 | 1 | 0 | **0** | | 1 | 0 | 1 | 0 | 1 | 1 | 1 | **1** | | 0 | 1 | 0 | 0 | 1 | 0 | 1 | **1** | | 1 | 1 | 0 | 1 | 0 | 1 | 0 | **0** | | 1 | 1 | 1 | 0 | 0 | 1 | 1 | **1** | | 0 | 1 | 0 | 0 | 1 | 0 | 0 | **0** | | 1 | 1 | 0 | 1 | 0 | 1 | 1 | **1** | | **1** | **1** | **0** | **1** | **0** | **1** | **0** | **0** |   Bit de paritate orizontală  (längsparitätbit)  Bit de paritate globală  (ganzparitätbit)  Bit de paritate verticală  (querparitätbit)  Figura 1. *Bitul de paritate* (Das Paritätbit) | | |
| 2. Reprezentarea numerelor în calculator |  | 2. Darstellung der Zahlen in den Computer |
| Sistemele de numerație binare sunt utilizate în aproape toate sistemele numerice, inclusiv în aplicațiile de procesare a semnalelor digitale (DSP), în rețele și în sistemele de calcul. Înainte de a alege un sistem de numerație, este important să-i înțelegem avantajele și dezavantajele și, de asemenea, să știm cum să convertim numerele dintr-un sistem în altul.  În continuare vom descrie următoarele sisteme de numerotare, avantajele și dezavantajele fiecărui sistem și tranziția între diferite sisteme.   * Întregi fără semn * Complementul față de 2 * Fracționare fără semn * Fracționare cu semn în Complementul față de 2 * Cod Gray * Mărime și semn * Complementul față de 2 deplasat * Complementul față de 1 * Virgulă mobilă |  | Binäre Zahlensysteme werden in nahezu allen digitalen Systemen, einschließlich der digitalen Signalverarbeitung (DSP), Netzwerk- und Computer verwendet.  Bevor Sie sich für ein Nummerierungssystem entscheiden, müssen Sie die Vor- und Nachteile der einzelnen Systeme verstehen.  Nachfolgend werden die folgenden Zahlensysteme beschrieben, die Vor- und Nachteile der einzelnen Systeme sowie die Umstellung zwischen verschiedenen Systemen.   * vorzeichenlose Ganzzahl * 2-Komplement Ganzzahl * Vorzeichenlose Bruchzahlen * 2-Komplement vorzeichen Bruchzahlen * Gray-Code * vorzeichenbehaftete Größenordnung * Offset von 2-Komplement * Einerkomplement * Fließkomma |
| 2.1 Întregi fără semn (*Unsigned Integers*) |  | 2.1 Vorzeichenlose ganze Zahlen (*Unsigned Integers*) |
| Cel mai bine cunoscut sistem de numerație este cel numit „Întregi fără semn“. La fel ca în sistemul zecimal, în Întregi fără semn se folosește o regulă simplă, de natură binară, de asociere a valorilor cifrelor. **Poziția unei cifre binare determină valoarea sa (adică valoarea asociată poziției unei cifre binare este 2*Poziție*)**. Această metodă de reprezentare este exact ca sistemul zecimal, unde valoarea asociată poziției unei cifre zecimale este 10*Poziție*. |  | Das bekannteste Nummerierungssystem ist die vorzeichenlose Ganzzahldarstellung. Wie das Dezimalsystem verwenden vorzeichenlose Ganzzahlen einen einfachen binären Platzwert. **Die Position einer Ziffer bestimmt ihren Wert (d. H. Der Platzwert einer Ziffer ist 2*Position*).** Diese Darstellung entspricht genau dem Dezimalsystem, bei dem der Platzwert 10*Position* ist. |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | **Ponderea (valoarea asociată poziției) - (*Platzwert*)** | 24 | 23 | 22 | 21 | 20 | | **Poziția (*Position*)** | 4 | 3 | 2 | 1 | 0 | | **Numărul binar (*Binäre Zahl*)** | 1 | 0 | 0 | 1 | 1 |   **Figura 2.** *Poziția biților în sistemul de numerație Întregi fără semn (*Bit Position in dem Vorzeichenlose ganze Nummerierungssystem*)* | | |
| **Tabelul 4.** *Conversia numerelor Întregi fără semn* (Die Umwandlung von Ganzzahlen ohne Vorzeichen)   |  |  |  | | --- | --- | --- | | Întregi fără semn  (*Ganzzahlen ohne Vorzeichen*) | Valoarea zecimală  (*Dezimalwert*) | Conversie  (*Umwandlung*) | | 01000 | 8 | 0+23+0+0+0=8 | | 10011 | 19 | 24+0+0+21+20=16+2+1=19 | | 11011 | 27 | 24+23+0+21+20=16+8+2+1=27 | | | |
| Se pot realiza cu ușurință operații aritmetice pe Întregi fără semn, respectând aceleași reguli ca în cazul numerelor zecimale. Dar, pentru numerele binare, cifra este transportată (*carry*) după 1 și nu după 9 (adică atunci când se adună două cifre 1, se va scrie un 0 pe poziția corespunzătoare și un 1 se va transportat către poziția următoare). Figura 3 prezintă modul de realizare a adunării a două numere Întregi fără semn. |  | Sie können arithmetische Operationen mit vorzeichenlosen Zahlen problemlos durchführen, indem Sie dieselben Regeln wie für Dezimalzahlen verwenden. Bei Binärzahlen wird die Ziffer jedoch nach 1 statt nach 9 übertragen (d. H., Wenn zwei 1en addiert werden, wird eine 0 an die entsprechende Position gesetzt und eine 1 wird zur nächsten Position befördert). Figura 3 zeigt, wie zwei vorzeichenlose Ganzzahlen zusammengefügt werden. |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | 11011 |  |  | 27 | | + | 10011 | = | + | 19 | | 1 | 01110 |  |  | 46 |   **Figura 3.** *Adunarea numerelor Întregi fără semn (*Addierte Ganzzahl ohne Vorzeichen*)* | | |
| Sistemul de numerație Întregi fără semn este foarte folosit. Principala limitare a acestui sistem de numerație constă în aceea că nu se pot reprezenta decât numerele cuprinse între 0 și (2*N*-1). Majoritatea sistemelor de prelucrare a semnalelor digitale au nevoie să stocheze atât numere pozitive, cât și numere negative. |  | Das vorzeichenlose Integer-Nummerierungssystem ist weit verbreitet. Die Hauptbeschränkung dieses Zahlensystems besteht darin, dass es nur die Zahlen im Bereich von 0 bis (2*N*-1) speichern kann. Die meisten Signalverarbeitungssysteme müssen sowohl positive als auch negative Zahlen speichern. |
| 2.2 Complementul lui 2 (*Two’s Complement*) |  | 2.2 2-Komplement Ganzzahl (*Two’s Complement*) |
| Cel mai folosit sistem de numerație care permite reprezentarea atât a numerelor pozitive, cât și a numerelor negative, este *Complementul lui 2*. Acest sistem de numerație este similar sistemului de numerație Întregi fără semn, cu excepția faptului că semnul bitului cel mai semnificativ (MSB) este negat. Pentru un număr pe *N* biți, bitul 0 are tot ponderea 20, bitul 1 are tot ponderea 21, bitul *N*-2 are tot ponderea 2*N*-2 , dar bitul *N*-1 (adică MSB-ul) are ponderea -2(*N*-1). Tabelul 5 prezintă valoarea zecimală corespunzătoare fiecărei poziții în cazul unei reprezentări în Complementul lui 2 pe 5 biți. |  | Das am häufigsten verwendete Nummerierungssystem, das sowohl positive als auch negative Zahlen speichern kann, ist die Zweierkomplement-Ganzzahl. Dieses System ist einer vorzeichenlosen Ganzzahl ähnlich, es sei denn, das Vorzeichen des höchstwertigen Bits (MSB) ist negiert. Für eine *N*-Bit-Nummer hat beispielsweise Bit 0 einen Wert von 20, Bit 1 hat einen Wert von 21, Bit *N*-2 hat einen Wert von 2*N*-2 und Bit *N*-1 (d. H. das MSB) hat einen Wert von -2(*N*-1). **Tabelul 5** zeigt den Dezimalwert für jede Position in einer 5-Bit-Zweierkomplement-Ganzzahl. |
| **Tabelul 5.** *Valorile întregi ale ponderilor biților în cazul reprezentării în Complementul lui 2 a unui număr pe 5 biți* (Zweierkomplement Integer-Werte für eine 5-Bit-Zahl)   |  |  |  | | --- | --- | --- | | **Poziție (*Position*)** | **Ponderea poziției (*Stellenwert*)** | **Valoarea zecimală (*Dezimalwert*)** | | 0 (LSB) | 20 | 1 | | 1 | 21 | 2 | | 2 | 22 | 4 | | 3 | 23 | 8 | | 4 (MSB) | -24 | -16 | | | |
| Câteva exemple: |  | Einige Beispiele: |
| 01000 = -24 × 0 + 23 × 1 + 22 × 0 + 21 × 0 + 20 × 0 = +8  11000 = -24 × 1 + 23 × 1 + 22 × 0 + 21 × 0 + 20 × 0 = -8  10000 = -24 × 1 + 23 × 0 + 22 × 0 + 21 × 0 + 20 × 0 = -16  10111 = -24 × 1 + 23 × 0 + 22 × 1 + 21 × 1 + 20 × 1 = -9 | | |
| Complementul lui 2 permite reprezentarea numerelor cuprinse între -2(*N*-1) și 2(*N*-1) -1. Pentru a obține complementul unui număr în acest sistem de numerație, se inversează toți biții și se adaugă 1. Următorii pași ilustrează cu titlu de exemplu modul în care se obține complementul numărului 9.  1. Se reprezintă numărul în Complementul lui 2: 9 = 01001  2. Se inversează toți biții: 10110  3. Se adună 1: (10110 + 1) = 10111 |  | Zweierkomplement-Ganzzahlen repräsentieren Zahlen im Bereich von -2(*N*-1) bis 2(*N*-1) -1. Um eine Zweierkomplement-Ganzzahl zu negieren, invertieren Sie einfach die Bits und fügen 1 hinzu. Die folgenden Schritte zeigen beispielsweise, wie die Zahl 9 zu -9 negiert wird.  1. Ersetzen Sie die binären Werte der Dezimalzahl: 9 = 01001  2. Invertieren Sie die Bits: 10110  3. Hinzufügen: (10110+1) = 10111 |
| Cel mai mare avantaj al sistemelor de numerație bazate pe Complementul lui 2 constă în aceea că adunarea și scăderea este identică cu adunarea și scăderea numerelor Întregi fără semn. Trebuie însă efectuată extensia de semn înainte de realizarea operației, iar transportul de la nivelul bitului cel mai semnificativ trebuie ignorat. Figura 4 prezintă câteva cazuri de adunare a numerelor reprezentate în Complementul lui 2. |  | Der größte Vorteil des Zweierkomplement-Zahlensystems besteht darin, dass das Addieren und Subtrahieren von Zweierkomplement Nummern dem Hinzufügen und Subtrahieren von vorzeichenlosen Zahlen entspricht. Die Zeichen Erweiterung muss jedoch vor der Operation ausgeführt werden, und die Ausführung des Addierers sollte ignoriert werden. Figura 4 zeigt, wie zwei 2-Bit-Zweierkomplementzahlen hinzugefügt werden. |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | **Pozitiv + pozitiv** | | | | |  | **Negativ + pozitiv** | | | | |  | **Negativ + negativ** | | | | | |  | 1 |  |  | 001 |  |  | -1 |  |  | 111 |  |  | -1 |  |  | 111 | | + | 1 |  | + | 001 |  | + | 1 |  | + | 001 |  | + | -1 |  | + | 111 | |  | 2 |  |  | 010 |  |  | 0 |  | 1 | 000 |  |  | -2 |  | 1 | 110 |   Figura 4. *Adunarea numerelor în Complementul lui 2 (*Zweierkomplement-Integeraddition*)* | | |
| Observații  1. Bitul de transport de la nivelul MSB se ignoră.  2. Toate numerele pozitive au MSB = 0, iar toate numerele negative au MSB = 1. De aceea, MSB indică semnul.  3. Dacă cele două numere care trebuie adunate nu sunt reprezentate pe același număr de biți, operația de adunare se va face pe lățimea celui mai mare dintre ele. La nivelul numărului mai mic este atunci necesar să se realizeze operațiunea de *extensie de semn*, care constă în replicarea MSB-ului până când se atinge o lățime egală cu cea a celuilalt operand. *Extensia de semn* nu modifică valoarea numărului! |  | Bemerkungen 1. Transportbit auf MSB-Ebene wird ignoriert. 2. Alle positiven Zahlen haben MSB = 0 und alle negativen Zahlen haben MSB = 1. Daher gibt der MSB das Vorzeichen an. 3. Wenn die zwei zu sammelnden Zahlen nicht mit der gleichen Anzahl von Bits dargestellt sind, wird der Zusammensetzungsvorgang mit der breitesten von ihnen ausgeführt. Bei der niedrigeren Zahl muss dann die *Vorzeichenerweiterungsoperation* ausgeführt werden, die darin besteht, das MSB zu replizieren, bis eine Breite erreicht ist, die der des anderen Operanden entspricht. Die *Vorzeichenerweiterung* ändert den Wert der Zahl nicht! |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 00001 |  | + 1 |  |  | 11000 |  | - 8 | | + | 10010 |  | -14 |  | + | 01110 |  | +14 | |  | 10011 |  | -13 |  | 1 | 00110 |  | +6 |   Extensie de semn (pozitiv)  *Vorzeichenerweiterung*  Extensie de semn (negativ)  *Vorzeichenerweiterung*  Bitul de transport de la MSB se ignoră!  Figura 5. *Extensia de semn pentru adunarea numerelor în Complementul lui 2 (*Vorzeichenerweiterungfür Zweierkomplement-Integeraddition*)* | | |
| 4. Dacă la Întregi cu semn nu se punea problema depășirii (*overflow*), la Complementul lui 2 există posibilitatea ca rezultatul adunării să fie un număr în afara intervalului de reprezentare. Depășirea se detectează astfel: dacă biții de transport de la nivelul bitului MSB și al bitului alăturat sunt diferiți, atunci avem depășire. |  | Wenn der Überlauf mit dem Vorzeichen nicht aufgetreten ist, besteht in 2-Komplement Ganzzahl die Möglichkeit, dass das Ergebnis der Assembly eine Zahl außerhalb des Darstellungsbereichs ist. Das Überholen wird wie folgt erkannt: Wenn die Transportbits vom MSB-Bit und den benachbarten Bits verschieden sind, haben wir überschritten. |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 10000 |  | -16 |  |  | 10100 |  | -12 |  |  | 00100 |  | + 4 | | + | 10010 |  | -14 |  | + | 11110 |  | - 2 |  | + | 10110 |  | - 10 | | 1 | 00010 |  | -30 |  | 1 | 10010 |  | -14 |  |  | 11010 |  | - 6 |   0  1  Biţi de transport diferiţi : OVERFLOW!  1  1  Biţi de transport identici : NU APARE (*kein*) OVERFLOW  0  0  Biţi de transport identici : NU APARE (*kein*) OVERFLOW  Figura 6. *Depășirea la adunarea numerelor în Complementul lui 2 (*Überlauf für Zweierkomplement-Integeraddition*)* | | |
| 2.3 Fracționare fără semn (*Unsigned fractional*) |  | 2.3 Vorzeichenlose Bruchzahlen (*Unsigned fractional*) |
| În DSP (*Digital Signal Processing*) și alte sisteme, este adeseori necesar să se stocheze numere care au atât o parte întreagă, cât și o parte fracționară. Întrucât anumite poziții binare pot fi negative, acest sistem de numerație poate stoca atât numere supraunitare cât și numere subunitare. Ponderea unei cifre în Fracționare fără semn este 2*Poziție*, unde poziția poate fi atât pozitivă, cât și negativă (vezi Figura 7). De aceea, sistemul de numerație Fracționare fără semn este un supraset al sistemului de numerație Întregi fără semn.  **Tabelul 6.** *Ponderile biților în cazul reprezentării în Fracționare fără semn a unui număr pe 5 biți* (Vorzeichenlose gebrochene Werte)**Tabelul 6** prezintă ponderile biților în cazul reprezentării în sistemul de numerație Fracționare fără semn a unui număr pe 5 biți, iar  **Tabelul 7** prezintă câteva exemple. |  | In DSP (Digital Verarbeitungssystem) und anderen Systemen ist es häufig erforderlich, Zahlen zu speichern, die sowohl eine Ganzzahl- als auch eine Bruch-komponente enthalten. Da einige Bitpositionen negativ sein können, kann das vorzeichenbehaftete fraktale System Zahlen speichern, die größer und kleiner als 1 sind. Der Stellenwert einer Ziffer im vorzeichenlosen Nummernsystem ist 2*Position*, wobei die Position positiv oder negativ sein kann (siehe Figura 7). Daher ist das vorzeichenlose fraktale Nummerierungssystem eine Obermenge des vorzeichenlosen ganzzahligen Nummerierungssystems.  **Tabelul 6** zeigt den Dezimalwert für jedes Bit in einer vorzeichenlosen gebrochenen Zahl.  **Tabelul 7** zeigt, wie der Dezimalwert einer vorzeichenbehafteten gebrochenen Zahl bestimmt wird. |
| |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | Ponderea (valoarea asociată poziției) - (*Platzwert*) | 22 | 21 | 20 | 2-1 | 2-2 | | Poziția (*Position*) | 2 | 1 | 0 | -1 | -2 | | Numărul binar (*Binäre Zahl*) | 1 | 0 | 0 | 1 | 1 |   Figura 7. *Poziția biților în sistemul de numerație Fracționare fără semn (*Bit Position in dem Vorzeichenlose Bruchzahlen*)* | | |
| **Tabelul 6.** *Ponderile biților în cazul reprezentării în Fracționare fără semn a unui număr pe 5 biți* (Vorzeichenlose gebrochene Werte)   |  |  |  | | --- | --- | --- | | **Poziția (*Position*)** | **Valoarea (*Stellenwert*)** | **Valoarea în zecimal (*Dezimalwert*)** | | -2 | 2-2 | 0.25 | | -1 | 2-1 | 0.50 | | 0 | 20 | 1 | | 1 | 21 | 2 | | 2 | 22 | 4 |     **Tabelul 7.** *Transformarea numerelor reprezentate în Fracționare fără semn în zecimal* (Vorzeichenlose fraktionale Umwandlung)   |  |  |  | | --- | --- | --- | | **Număr în Fracționare fără semn**  **(*Vorzeichenlose gebrochene Zahl*)** | **Valoarea în zecimal (*Dezimalwert*)** | **Valoarea după conversia în zecimal (*Umwandlung*)** | | 01001 | 2.25 | 0 + 21 + 0 + 0 + 2-2 = 2 + 0.25 = 2.25 | | 11011 | 6.75 | 22 + 21 + 0 + 2-1 + 2-2 = 4 + 2 + 0.5 + 0.25 = 6.75 | | 00010 | 0.5 | 0 + 0 + 0 + 2-1 + 0 = 0.5 | | | |
| Sistemul de numerație Fracționare fără semn folosește o notație convenabilă, pentru a urmări poziția virgulei (atât în binar, cât și în zecimal). Un număr care are *N* biți la stânga virgulei și *M* biți la dreapta acesteia este un număr *N*.*M* (de exemplu, dacă are 8 biți la stânga virgulei și 3 biți la dreapta, se spune că este un număr 8.3).  Dacă toate numerele din sistemul de numerație au aceeași valoare a lui *M* (adică partea fracționară este reprezentată pe același număr de biți), atunci operațiile aritmetice sunt foarte simplu de realizat. De exemplu, se poate aduna un număr 12.3 cu un număr 8.3 folosind un circuit de adunare pe 15 biți. În **Figura 8** se prezintă modul în care se realizează adunarea numerelor 00100101.101 (37.625) și 001101110011.001 (883.125). |  | Vorzeichenlose gebrochene Zahlen verwenden eine bequeme Notation, um die Position eines Radixpunkts (d. h. des Binär- oder Dezimalpunkts) zu verfolgen; Eine Zahl mit *N* Bits links vom Radixpunkt und *M* Bits rechts ist eine *N*.*M*-Nummer (z. B. hat eine 8.3-Nummer 8 Ziffern links vom Radixpunkt und 3 Ziffern rechts).  Wenn alle Zahlen in Ihrem System den gleichen *M*-Wert haben (d. H. Dieselbe gebrochene Bitbreite), sind arithmetische Operationen unkompliziert. Beispielsweise können Sie einer 12.3-Nummer eine 8.3-Nummer hinzufügen, indem Sie einen 15-Bit-Binäraddierer verwenden. **Figura 8** zeigt, wie 00100101.101 (37.625) zu 001101110011.001 (883.125) hinzugefügt wird. |
| |  |  | | --- | --- | | 37.625 | 000000100101.101 | | +883.125 | +001101110011.001 | | 920.750 | 001110011000.110 |   **Figura 8.** *Adunarea în cazul numerelor cu aceeași valoare a lui* M *(*Addition mit den gleichen *M*-Werten*)* | | |
| Se observă că este necesar să se completeze cu 4 biți de 0, pentru ca virgula să ajungă pe aceeași poziție. |  | Beachten Sie, dass Sie 4 Bits mit 0 eingeben müssen, damit das Komma an die gleiche Stelle kommt. |
| În cazul adunării unor numere cu valori diferite ale lui *M*, trebuie completat cu zerouri pentru a păstra virgulele aliniate. De exemplu, dacă se adună un număr 8.3 cu un număr 6.5, numărul 8.3 trebuie completat cu zerouri astfel încât să devină un număr 8.5. astfel, trebuie folosit un circuit sumator pe 13 biți în locul unuia pe 11 biți. În **Figura 8** se prezintă modul în care se realizează adunarea numerelor 11011011.110 (219.750) și 110111.11011 (55.84375). |  | Um Zahlen mit unterschiedlichen *M*-Werten hinzuzufügen, müssen Sie zusätzliche Nullen hinzufügen, damit die Radixpunkte ausgerichtet bleiben. Um beispielsweise eine 8.3-Nummer zu einer 6.5-Nummer hinzuzufügen, müssen Sie die 8.3-Nummer mit Nullen auffüllen, um eine 8.5-Nummer zu erstellen. Daher müssen Sie einen 13-Bit-Nummernaddierer anstelle eines 11-Bit-Nummernaddierers verwenden, um die beiden Zahlen zu addieren. Abbildung 6 zeigt, wie 11011011.110 (219.750) zu 110111.11011 (55.84375) hinzugefügt wird. |
| |  |  | | --- | --- | | 219.75000 | 11011011.11000 | | +055.84375 | * 00110111.11011 | | 275.59375 | 100010011.10011 |   Figura 9. *Adunarea în cazul numerelor Fracționare fără semn cu valori diferite ale lui* M*. Biții introduși în completare sunt evidențiați cu gri (*Addieren mit unterschiedlichen *M*-Werten Vorzeichenlose Bruchzahlen. Die für das Auffüllen verwendeten Ziffern werden grau hervorgehoben*)* | | |
| În afară de alinierea virgulelor zecimale, nu există diferențe între sistemele de numerație Întregi fără semn și Fracționare fără semn. De fapt, un număr Întreg fără semn este pur și simplu un număr Fracționar fără semn de forma *N*.0 (adică Întregii fără semn nu au deloc biți în partea fracționară). Orice circuit hardware va funcționa atât pentru Întregi fără semn, cât și pentru Fracționare fără semn. |  | Abgesehen vom Ausrichten der Radixpunkte gibt es keinen Unterschied zwischen den vorzeichenlosen Ganzzahlsystemen und den vorzeichenlosen gebrochenen Nummerierungssystemen. Tatsächlich ist eine vorzeichenlose Ganzzahl einfach eine vorzeichenlose *N*.0-Bruchzahl (d. H. Vorzeichenlose Ganzzahlen haben keine Bits rechts vom Radixpunkt). Jede Hardware, die für ganzzahlige Zahlen erstellt wurde, arbeitet mit gebrochenen Zahlen. |
| 2.4 Fracționare în complementul lui 2 (*Two’s Complement signed fractional*) |  | 2.4 2-Komplement vorzeichen Bruchzahlen (*Two’s Complement signed fractional*) |
| Acest sistem de numerație este similar cu sistemul de numerație Fracționare fără semn, în sensul că are o notație de tipul *N*.*M* și virgulele trebuie aliniate în timpul operațiilor aritmetice.    Figura 10 prezintă cum se realizează adunarea unui număr 7.5 cu un număr 9.2 în acest sistem de numerație. |  | Wie das vorzeichenlose gebrochene Nummerierungssystem verwendet das mit zwei-Komplementen versehene gebrochene Nummerierungssystem eine *N*.*M*-Notation, und die Radixpunkte müssen während arithmetischer Operationen ausgerichtet sein.  Abbildung 10 zeigt, wie eine 8.3-Nummer zu einer 5.5-Nummer hinzugefügt wird. |
| |  |  | | --- | --- | | -36.28125 | 111011011.10111 | | * 154.25000 | * 010011010.01000 | | +117.96875 | 001110101.11111 |   Figura 10. *Adunarea numerelor Fracționare în complementul lui 2 cu valori diferite ale lui* M*. Biții introduși în completare sunt evidențiați cu gri (*Zweierkomplementierte fraktionierte Addition. Die für das Auffüllen verwendeten Ziffern werden grau hervorgehoben.*)* | | |
| 2.5 Codul Gray |  | 2.5 Gray Code |
| Codul Gray este un sistem de numerație care se folosește adeseori în domeniul aplicațiilor bazate pe senzori. Caracteristica principală a codului Gray este aceea că pe măsură ce înaintăm, un singur bit se modifică de-a lungul cuvintelor de cod.  Pentru a obține cuvintele codului Gray pe 2 biți începem de la cuvintele codului Gray pe 1 bit: |  | Gray Code ist ein Nummerierungssystem, das hauptsächlich in realen Sensoranwendungen verwendet wird. Das grundlegende Merkmal von Gray-Code ist, dass sich jeweils nur ein Bit ändert, während Sie die Zahlen nacheinander durchlaufen.  Für den Wörtern des 2-Bit-Gray zu erhalten, beginnen wir das Gray-Codewort mit 1 Bit: |
| 0  1 | | |
| Pe doi biți vor fi patru cuvinte de cod binare. Primele două cuvinte de cod binare ale codului Gray pe 2 biți se obțin pur și simplu concatenând un ‘0’ (pe poziția MSB) la cuvintele de cod Gray de 1 bit, după cum urmează (în *italice*): |  | Zwei Bits sind vier binäre Codewörter. Die ersten beiden binären Codewörter des 2-Bit-Gray-Codes werden einfach durch Verketten einer '0' (an der MSB-Position) mit den 1-Bit-Gray-Codewörtern wie folgt erhalten (in Kursivschrift): |
| *0*0  *0*1 | | |
| Ultimele două cuvinte de cod binare ale codului Gray pe 2 biți au un ‘1’ pe poziția MSB. Apoi, ne imaginăm că există o oglindă care separată primele două cuvinte binare de ultimele. Cuvintele binare ale codului Gray pe 1 bit vor fi apoi reflectate în această oglindă, iar aceste reflexii se supun legilor fizicii: cuvintele care sunt mai aproape de oglindă apar ca fiind mai apropiate de suprafața ei, iar cuvintele care sunt mai îndepărtate vor părea că se reflectă mai adânc în oglindă: |  | Die letzten beiden Binärcodes des 2-Bit-Gray-Codes haben an der MSB-Position eine '1'. Dann stellen wir uns vor, dass es einen Spiegel gibt, der die ersten beiden binären Wörter von den letzten trennt. Die Binärwörter des 1-Bit-Gray-Codes werden dann in diesem Spiegel reflektiert, und diese Reflexionen unterliegen den Gesetzen der Physik: Die Wörter, die dem Spiegel am nächsten liegen, erscheinen näher an seiner Oberfläche, und die weiter entfernten Wörter scheinen zu sein spiegelt sich tiefer im Spiegel: |
| *0*0  *Spiegel*  *0*1  *1*0  *1*1 | | |
| Cuvintele codului Gray pe 3 biți se vor obține în aceeași manieră, pe baza Cuvintele codului Gray pe 2 biți: |  | Die Wörter des 3-Bit-Gray-Codes werden auf die gleiche Weise erhalten, basierend auf den 2-Bit-Gray-Wortwörtern: |
| *0*00  001  *0*10  *Spiegel*  *0*11  *1*11  *1*10  *1*01  *1*00 | | |
| Cuvintele codului Gray pe 4 biți se vor obține în aceeași manieră, pe baza Cuvintele codului Gray pe 3 biți: |  | Die Wörter des 3-Bit-Gray-Codes werden auf die gleiche Weise erhalten, basierend auf den 2-Bit-Gray-Wortwörtern: |
| *0*000  *0*001  *0*010  *0*011  *0*111  *0*110  *0*101  *Spiegel*  *0*100  *1*100  *1*101  *1*110  *1*111  *1*011  *1*010  *1*001  *1*000 | | |
| 2.6 Mărime și semn (*Signed magnitude*) |  | 2.6 Vorzeichenbehaftete Größenordnung (*Signed magnitude*) |
| Numerele reprezentate în Mărime și semn sunt utile în aplicații în care semnul și mărimea unui număr trebuie stocate separat. În acest sistem de numerație MSB reprezintă semnul numărului (0 = pozitiv, 1 = negativ) iar toți ceilalți biți reprezintă mărimea. Această notație este similară cu notația zecimală, unde se utilizează semnele + și – ca indicatori ai semnului și restul biților se utilizează pentru reprezentarea mărimii.  Sistemele de numerație Mărime și semn și Complementul lui 2 folosesc ambele MSB pentru a determina semnul. Însă ele nu trebuie confundate. Deși în MSB poate fi utilizat pentru a determina semnul, ceilalți biți nu reprezintă mărimea, atunci când semnul este negativ. În plus, Complementul lui 2 nu are decât o reprezentare a lui zero, în timp ce sistemul de numerație Mărime și semn are două (+0 și -0).  Pentru a efectua operații aritmetice complexe pe date reprezentate în Mărime și semn, de regulă cel mai ușor este ca numerele să fie convertite în Complementul lui 2, ca operația să fie efectuată și apoi numerele să fie convertite înapoi în Mărime și semn. |  | Vorzeichenbehaftete Größen sind nützlich für Anwendungen, bei denen auf Vorzeichen und Größe einer Zahl separat zugegriffen werden muss. In Systemen mit vorzeichenbehafteten Beträgen repräsentiert das MSB das Vorzeichen der Zahl (d. H. 0 = positiv, 1 = negativ) und alle anderen Bits repräsentieren die Größe. Diese Schreibweise ähnelt der dezimalen Schreibweise, die ein + und - als Vorzeichenbit verwendet und die verbleibenden Bits zur Darstellung der Größe verwendet.  Das vorzeichenbehaftete und das Zweierkomplement-Nummerierungssystem verwenden beide das MSB, um das Vorzeichen zu bestimmen. Verwechseln Sie die vorzeichenbehaftete Größe jedoch nicht mit Zweierkomplement. Obwohl das MSB zur Bestimmung des Vorzeichens in Zweierkomplement verwendet werden kann, repräsentieren die anderen Bits nicht die Größe, wenn das Vorzeichen negativ ist. Zusätzlich hat das Zweierkomplement nur eine Darstellung von Null, während das vorzeichenbehafteten Nummerierungssystem hat zwei (d.h. +0 und -0).  Um komplexe arithmetische Operationen mit vorzeichenbehafteten Betragsdaten durchzuführen, ist es in der Regel einfacher, die Zahlen in Zweierkomplement umzuwandeln, die Operationen auszuführen und die Zahlen dann gegebenenfalls wieder in ein vorzeichenbehaftetes Größenordnungsystem umzuwandeln. |
| 2.7 Complementul lui 2 cu deplasament (*Offset two’s Complement*) |  | 2.7 Offset Zweierkomplement (*Offset two’s Complement*) |
| Sistemul de numerație Complementul lui 2 cu deplasament este foarte folosit în convertoarele analog/digitale și digital /analoge. Caracteristica distinctivă constă în aceea că numerele de la -4 la 3 se succed monoton, așa cum numărăm în binar de la 000 la 111. Spre deosebire de aceasta, valorile zecimale ale numerelor reprezentate în Complementul lui 2 merg de la 0 la 3 și apoi de la -4 la -1 pe măsură ce numărul binar crește de la 000 la 111.  Pentru a converti numerele din Complementul lui 2 în Complementul lui 2 cu deplasament este suficient să inversăm MSB, după cum se vede în **Tabelul 8**. |  | Das Zwei-Offset-Komplement-Nummerierungssystem wird von vielen D / A- und A / D-Wandlern verwendet. Das Unterscheidungsmerkmal dieses Nummerierungssystems besteht darin, dass sich die Zahlen von -4 nach 3 monoton bewegen, wenn Sie binär von 000 bis 111 zählen - es gibt keine Sprünge oder Unterbrechungen. Im Gegensatz dazu geht der Dezimalwert der Zweierkomplementzahlen von 0 bis 3 und zählt dann von -4 bis -1, wenn die Binärzahl von 000 bis 111 fortschreitet.  Um Zahlen von Zweierkomplement in Offset-Zweierkomplement umzuwandeln, invertieren Sie einfach das MSB, wie in **Tabelul 8** gezeigt. |
| **Tabelul 8.** *Conversia numerelor în Complementul lui 2 cu deplasament* (Umwandeln des Offset-Zweierkomplement)   |  |  |  | | --- | --- | --- | | Zecimal (*Dezimalwert*) | Complementul lui 2  (*Zweierkomplement*) | Complementul lui 2 cu deplasament  (Offset-Zweierkomplement) | | -4 | 100 | 000 | | -3 | 101 | 001 | | -2 | 110 | 010 | | -1 | 111 | 011 | | 0 | 000 | 100 | | 1 | 001 | 101 | | 2 | 010 | 110 | | 3 | 011 | 111 | | | |
| 2.8 Complementul lui 1 (*One’s Complement*) |  | 2.8 Einerkomplement (*One’s Complement*) |
| Sistemul de numerație Complementul lui 1 este folosit mai rar, deoarece prezintă aceleași dezavantaje ca toate sistemele de numerație nebazate pe Complementul lui 2: este greu să se efectueze operații aritmetice cu el și are două reprezentări ale lui 0. Când adunăm un număr cu opusul lui în Complementul lui 1, rezultatul nu este 0, ceea ce generează o algebră inconsistentă. Există două reprezentări ale lui 0, 000 și 111, ceea ce fac mai dificilă detecția lui zero (trebuie realizată o operație SAU pe *N* biți și o operație ȘI pe *N* biți). Pentru a nega un număr reprezentat în Complementul lui 1 trebuie să inversăm toți biții (nu se mai adună 1 ca în cazul Complementului lui 2). |  | Das Einerkomplement-Nummerierungssystem wird selten verwendet, da es die gleichen Nachteile wie alle Nicht-Zweierkomplement-Nummern hat - es ist schwierig, arithmetische Operationen auszuführen, und es hat zwei Repräsentationen von Null (d. H. +0 und -0).  Wenn Sie zu ihrer Umkehrung eine Ein-Komplement-Nummer hinzufügen, ist die Antwort nicht Null, wodurch eine inkonsistente Algebra entsteht. Es gibt beiden Darstellungen von Null 000 und 111, was die Nullpunkterfassung schwieriger macht (d. H. Sie müssen ein *N*-Bit-ODER und ein *N*-Bit-UND-Gatter verwenden). Um eine Zahl in dem EinerKomplement-Nummerierungssystem zu negieren, invertieren Sie einfach alle Bits (d. H. Sie fügen keine 1-Zahl wie in Zweierkomplement hinzu). |
| 2.9 Virgulă mobilă (*Floating point*) |  | 2.9 Gleitkomma-Zahlen (*Floating point*) |
| Reprezentarea numerelor în virgulă mobilă este adeseori necesară în domeniul științific și tehnic mai ales pentru numere mari. În limbajele de programare este pus la dispoziția programatorului tipul de date REAL. Însă această denumire este înșelătoare, deoarece aritmetica în virgulă mobilă permite cel mult reprezentarea aproximativă a numerelor reale. Pentru o mai bună evaluare a limitelor și posibilităților reprezentărilor bazate pe virgula mobilă prezentăm o serie de aspecte legate de reprezentarea numerelor conform standardelor IEEE.  Reprezentarea numerelor în virgulă mobilă *Z* se bazează pe așa-numita **reprezentare semi-logaritmică**, des întâlnită în domeniul tehnic, în care *mantisa* și *exponentul* sunt codificate ca vectori de biți: |  | Gleitkomma-Zahlen werden im wissenschaftlich-technischen Bereich insbesondere bei großen Zahlenbereichen vielfach benötigt. In den Programmiersprachen wird dem Programmierer der REAL-Datentyp zur Verfügung gestellt. Diese Bezeichnung ist aber irreführend, denn die verfügbare Gleitkomma-Arithmetik erlaubt höchstens die näherungsweise Darstellung von reellen Zahlen. Zwecks besserer Einschätzung der Grenzen und Möglichkeiten üblicher Gleitkomma-Darstellungen wollen wir hier v.a. auf die Darstellung nach IEEE-Standards eingehen.  Die Darstellung von Gleitkomma-Zahlen *Z* basiert auf der im technischen Bereich üblichen **halblogarithmischen Darstellung**, deren Mantisse und Exponent als Bitvektoren kodiert werden: |
|  | | |
| Formatele cele mai comune sunt următoarele: |  | Übliche Formate sind die folgenden: |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | a) | *VZM* | *VZE* | *E* | *MANTISSENBETRAG M* | | | |
| Unde: *VZM = Bitul de semn al mantisei M*; *VZE = Bitul de semn al exponentului E*. |  | Mit: *VZM = Vorzeichen zu M*; *VZE = Vorzeichen zu E*. |
| |  |  |  |  | | --- | --- | --- | --- | | b) | *VZM* | *CHARAKTERISTIK C* | *MANTISSENBETRAG M* | | | |
| *Caracteristica* se calculează adăugând magnitudinea celui mai mic exponent dorit la toți exponenții.  Acest lucru permite practic lucrul cu exponenți exclusiv pozitivi. Astfel se realizează simplificarea multor algoritmi. Numai în momentul conversiei în reprezentarea externă se scade din nou numărul.  Standardul IEEE de reprezentare a numerelor reale propune trei moduri de reprezentare a numerelor reale:  - Formatul scurt pe 4 octeți (*simplă precizie*)  - Formatul lung pe 8 octeți (*dublă precizie*)  - Formatul temporar pe 10 octeți (pentru coprocesorul matematic).  Bitul de semn *VZM* se reprezintă pe un singur bit. Caracteristica se reprezintă de regulă pe 8, 11, sau 15 biți la formatul scurt, lung, respectiv temporar. Mantisa se reprezintă de regulă pe 23, 52, respectiv 64 de biți.  Pentru fiecare reprezentare: *VZM* este 0 dacă numărul este pozitiv și 1 dacă numărul este negativ. Caracteristica *C* = *E* + 7F16 (respectiv 3FF16 la dublă precizie și 3FFF16 la formatul temporar).  Pentru găsirea mantisei mai întâi se normalizează numărul scris în baza 2, adică se scrie numărul sub forma: NR = 1.< alte cifre binare > × 2*E*.  La reprezentarea în format IEEE *simplă precizie* și *dublă precizie*, mantisa este formată din cifrele de după virgulă, deci primul 1 dinaintea virgulei nu se mai reprezintă în mantisă, iar la formatul temporar se reprezintă toate cifrele din număr.  Exemple  *Exemplul 1*  Să se reprezinte în format IEEE *simplă precizie* numărul (17.6)10.  Se va converti separat partea întreagă și cea zecimală și se obține:  Partea întreagă:  (17)10 = (11)16 = (0001 0001)2  Partea zecimală: (0.6)10 = (0.(1001))2 (se observă că numărul este periodic).  Deci (17.6)10= 10001.(1001)2.  Se normalizează numărul: (17.6)10 = 10001. (1001)2 = 1.0001(1001) × 24 (deși era mai  corect în loc de 24 să se scrie (10100)2 pentru că notarea era în baza 2; faptul că se calculează caracteristica mai ușor în hexazecimal decât în binar poate fi o scuză).  Din această reprezentare se poate deduce mantisa (ceea ce este după virgulă, deci fără acel 1 dinaintea virgulei care prin convenție nu se mai reprezintă) și anume:  *M* = 0001(1001)2.  În continuare se calculează caracteristica: |  | Die Charakteristik wird berechnet, indem der Betrag des kleinsten gewünschten Exponenten auf alle Exponenten addiert. Dies erlaubt praktisch ein Arbeiten mit ausschließlich positiven Exponenten. Es bewirkt eine Vereinfachung vieler Algorithmen. Erst bei Wandlung in die externe Darstellung wird die addierte Zahl wieder abgezogen.  Der IEEE-Standard für die Darstellung reeller Zahlen schlägt drei Möglichkeiten zur Darstellung reeller Zahlen vor:  - Kurzes (4-Byte-Format) – *einfache Präzision*  - Langes (8-Byte-Format) – *doppelte Präzision*  - Temporäres 10-Byte-Format (für den mathematischen Coprozessor).  Das Vorzeichenbit *VZM* wird in einem einzelnen Bit dargestellt. Die Charakteristik ist normalerweise 8, 11 oder 15 Bit in Kurz-, Lang- oder temporärem Format. Mantissa ist in der Regel 23, 52 und 64 Bit normalerweise.  Für jede Darstellung: *VZM* ist 0, wenn die Zahl positiv ist, und 1, wenn die Zahl negativ ist. Die Charakteristik *C* = *E* + 7F16 (3FF16 für doppelte Präzision und 3FFF16 für temporäres Format).  Um die Mantisse zu finden, zuerst wird die in die Basis 2 geschriebene Zahl normalisiert, d.h. die Zahl wird in der folgenden Form geschrieben: NR = 1. <andere Binärziffern> × 2*E*.  Bei der Darstellung im IEEE *einfache Präzision* und *doppelte Präzision* besteht die Mantisse nachher die Kommas nummerierten Ziffern, sodass die erste 1 vor dem Komma nicht mehr in der Mantisse enthalten sind. Im temporären Format alle Zahlen in der Zahl enthalten sind.  Beispiele  *Beispiel 1*  Stellen Sie in einem kurzen IEEE *einfache Präzision* Format die Nummer (17.6)10 dar.  Es wird das Ganze und den Dezimalteil getrennt konvertieren und erhalten:  Ganzer Teil:  (17)10 = (11)16 = (0001 0001)2  Dezimalteil: (0.6)10 = (0.(1001))2 (es wird angemerkt, dass die Zahl periodisch ist).  Also (17.6)10= 10001.(1001)2.  Normalisieren Sie die Zahl: (17.6)10 = 10001. (1001)2 = 1.0001(1001) × 24 (obwohl es mehr war richtig statt 24 schreiben (10100)2, weil die Note im Basis 2 war; Die Tatsache, dass die Charakteristik einfacher hexadezimal als binär zu berechnen ist, kann eine Entschuldigung sein).  Aus dieser Darstellung können wir die Mantisse ableiten (die nach dem Komma steht, d.h. ohne die 1 vor dem Komma, die laut Konvention nicht mehr steht), nämlich:  *M* = 0001(1001)2.  Als nächstes wird die Charakteristik berechnet: |
| *C* = *E* + 7F16 = 4 + 7F16 = 8316 = (1000 0011)2 | | |
| Se va scrie bitul semn 0 și deja se poate trece la scrierea reprezentării: |  | Es wird das Vorzeichenbit 0 geschrieben und Sie können bereits zum Schreiben der Darstellung wechseln: |
| |  |  |  | | --- | --- | --- | | Semn (*Vorzeichenbit*) | *Charakteristik* | *Mantissa* | | 0 | 10000011 | 0001100110011001100 | | | |
| Rezultatul final al reprezentării este: (41 8C CC CC)16.  În cazul practic, în memoria calculatorului, datorită unei rotunjiri care se face la ultimul  bit al reprezentării, se poate observa că în mantisă ar mai urma un 1 după cei 23 de biți, iar calculatorul va face rotunjire superioară, de aceea pe ultimul bit (LSB) va apărea 1, iar reprezentarea va fi: (41 8C CC CD)16.  *Exemplul 2*  În mod analog se va reprezenta (-23.5)10:  (23)10 = (17)16 = (1 0111)2  (0.5)10 = (0.1)2  Deci (23.5)10 = 10111.1 = 1.01111 × 24, de unde rezultă *M* = 0111100000000000… (23 de biți).  Caracteristica *C* = 7F16 + 416 = 8316.  Se pune bitul semn pe 1.  Reprezentarea direct în hexazecimal este C1 BC 00 0016.  În continuare se pune problema inversă reprezentării: se dă reprezentarea unui număr în format IEEE simplă precizie și se cere aflarea numărului real care este astfel reprezentat.  *Exemplul 3*  Se dă reprezentarea 43 04 33 3316 și se cere să se obțină valoarea zecimală a numărului real reprezentat.  Se scrie în binar reprezentarea: (0100 0011 0000 0100 0011 0011 0011 0011)2. De aici se deduce că:  - Semnul este 0.  - Caracteristica *C* = 1000 01102 = 8616.  Rezultă deci că exponentul *E* = 8616 - 7F16 = 716.  - Mantisa *M* = 0000 1000 0110 0110…  Numărul este:  Nr = 1.*M* × 2*E* =(1.0000 1000 0110…)2 × 27  = (1000 0100.00110011…)2  = 128 + 4 + 0.125 + 0.0625 + …  ≈ 132.1875  Valoarea exactă era 132.2. |  | Das Endergebnis der Darstellung ist: (41 8C CC CC)16.  In der Praxis im das Speicher von dem Computer, aufgrund der Rundung beim letzten Bit von der Darstellung, kann man sehen, dass die Mantisse ein 1 nach dem 23-Bit folgen würden und der Computer wird oben abgerundet sein, so das letzte Bit (LSB) wird 1 erscheinen, und die Darstellung wird: (41 8C CC CD)16.  *Beispiel 2*  Analog wird die Darstellung von (-23.5)10 sein:  (23)10 = (17)16 = (1 0111)2  (0.5)10 = (0.1)2  Also (23.5)10 = 10111.1 = 1.01111 × 24, was zu *M* = 0111100000000000… (23 Bits) führt.  Die Charakteristik *C* = 7F16 + 416 = 8316.  Setzen Sie das Vorzeichenbit auf 1.  Die direkte hexadezimale Darstellung ist C1 BC 00 0016.  Weitere umkehren die Frage der Darstellung: Darstellung gegeben wird eine Zahl in IEEE einfacher Präzision und die reelle Zahl erfordert zu finden ist so dargestellt.  *Beispiel 3*  Die Darstellung 43 04 33 3316 ist gegeben und es ist erforderlich, den Dezimalwert der dargestellten reellen Zahl zu erhalten.  Die Darstellung ist binär geschrieben: (0100 0011 0000 0100 0011 0011 0011 0011)2. Daraus folgt:  - Das Vorzeichen ist 0.  - Die Charakteristik *C* = 1000 01102 = 8616.  Daraus folgt der Exponent *E* = 8616 - 7F16 = 716.  - Die Mantissa *M* = 0000 1000 0110 0110…  Die Zahl ist:  Nr = 1.*M* × 2*E* = (1.0000 1000 0110…)2 × 27  = (1000 0100.00110011 ...)2  = 128 + 4 + 0,125 + 0,0625 + ...  ≈ 132,1875  Der genaue Wert war 132,2. |
| Utilizarea Caracteristicii prezintă, de asemenea, avantajul de a permite efectuarea comparațiilor de magnitudine pentru numerele în reprezentate în virgulă mobilă folosind exact comenzile corespunzătoare pentru compararea numerelor naturale sau reprezentate în Complementul lui 2. |  | Die Benutzung der Charakteristik hat weiter den angenehmen Effekt, dass betragmäßige Größenvergleichen für Gleitkommazahlen mit den entsprechenden Befehlen für natürliche oder 2er-Komplement-  Zahlen ausgeführt werden können. |
| 3. Reprezentări binare și ordini de plasare |  | 3. Binäre Darstellungen und Platzierungsaufträge |
| 3.1 Dimensiunea reprezentării |  | 3.1 Darstellungsdimension |
| Fiind vorba de calcule efectuate cu o mașină, există o serie de restricții legate de reprezentarea numerelor. Cea mai importantă dintre ele este *dimensiunea reprezentării*, adică numărul maxim de biți din reprezentarea unui număr. Să notăm cu *n* această dimensiune de reprezentare. Numărul *n* este o constantă a sistemului de calcul, stabilită la proiectarea acestuia.  Valorile cele mai uzuale ale lui *n* la sistemele de calcul actuale sunt: 8, 16, 32 și 64.  Tot în categoria restricțiilor intră și faptul că dimensiunile celor doi operanzi care participă la o operație, precum și dimensiunile rezultatului sunt de asemenea constante ale calculatorului, indiferent de tipul de codificare a numerelor. Pentru a preciza regulile de dimensionare în operațiile binare asupra numerelor întregi, vom folosi sintagma "operație pe *n* biți". Regulile de dimensionare în urma operațiilor sunt următoarele:  a) Operațiile de adunare pe *n* biți și scădere pe *n* biți presupun că ambii termeni sunt reprezentați pe câte *n* biți, iar rezultatul (suma sau diferența) se va reprezenta tot pe *n* biți, vezi Figura 11. |  | In Bezug auf Maschinenberechnungen gibt es eine Reihe von Einschränkungen bei der Darstellung von Zahlen. Das wichtigste davon ist *die Größe der Darstellung*, dh die maximale Anzahl von Bits in der Darstellung einer Zahl. Notieren wir mit *n* diesen diese Dimension der Repräsentation. Die Zahl *n* ist eine Konstante des Berechnungssystems, die bei seiner Auslegung festgelegt wurde.  Die häufigsten Werte von *n* in aktuellen Computersystemen sind: 8, 16, 32 und 64.  Ebenfalls in die Kategorie der Einschränkungen fällt die Tatsache, dass die Dimensionen der beiden an einer Operation beteiligten Operanden sowie die Größe des Ergebnisses unabhängig von der Art der Codierung der Zahlen auch für den Computer konstant sind. Um Größenregeln in Binäroperationen für Ganzzahlen festzulegen, verwenden wir den Ausdruck „*n*-Bit-Operation”. Die Größenregeln für folgende Vorgänge lauten wie folgt:  a) *N*-Bits-Addition und *N*-Bits-Subtraktionsoperationen setzen voraus, dass beide Terme durch *n* Bits dargestellt werden und das Ergebnis (Summe oder Differenz) auch durch *n* Bits dargestellt wird, siehe Figura 11. |
| Termen (*Term*) 1 (*n* biţi)  Termen (*Term*) 2 (*n* biţi)  +  Suma (*Summe*) (*n* biţi)  Termen (*Term*) 1 (*n* biţi)  Termen (*Term*) 2 (*n* biţi)  -  Diferenţa (*Differenz*) (*n* biţi)  Figura 11. *Dimensiuni de reprezentare la adunare și scădere* (Dimensionen der Darstellungfür die Addition und die Subtraktion) | | |
| b) Înmulțirea pe *n* biți presupune că ambii factori sunt reprezentați pe câte *n* biți, iar produsul lor va fi reprezentat pe 2 × *n* biți, vezi **Figura 12**. |  | b) *n*-Bits Multiplikation setzt voraus, dass beide Faktoren durch *n* Bits dargestellt werden und ihr Produkt durch 2 × *n* Bits dargestellt wird, siehe **Figura 12**. |
| Termen (*Term*) 1 (*n* biţi)  Termen (*Term*) 2 (*n* biţi)  ×  Produsul (*Produkt*) (2*n* biţi)  **Figura 12.** *Dimensiuni de reprezentare la înmulțire* (Dimensionen der DarstellungfürdieMultiplikation) | | |
| c) împărțirea pe *n* biți (oarecum invers față de înmulțire), impune condiția ca deîmpărțitul să fie reprezentat pe 2 × *n* biți, iar împărțitorul pe *n* biți. Operația furnizează două rezultate: câtul reprezentat pe *n* biți și restul reprezentat tot pe *n* biți, vezi **Figura 13**. |  | c) *n*-Bits Division (etwas entgegengesetzt zur Multiplikation) impliziert, dass der Dividend auf 2 × *n* Bits und der Divisor auf *n* Bits dargestellt wird. Die Operation liefert zwei Ergebnisse: den Quotient, der auf *n* Bits dargestellt ist, und den Rest, der ebenfalls auf *n* Bits dargestellt ist, siehe Figura 13. |
| Împărţitor (*Divisor*) (*n* biţi)  Cât (*Quotient*) 2 (*n* biţi)  Deîmpărţit (*Dividend*) (2*n* biţi)  Rest 2 (*n* biţi)  **Figura 13.** *Dimensiuni de reprezentare la împărțire* (Dimensionen der DarstellungfürdieDivision) | | |
| Dacă rezultatul unei operații nu încape în dimensiunea de reprezentare, atunci se vor pierde biții cei mai semnificativi, rămânând biții mai puțin semnificativi: 0, 1, 2 ș.a.m.d., calculatorul semnalând fenomenul de *depășire*. |  | Wenn das Ergebnis einer Operation nicht in die Repräsentationsdimension passt, gehen die höchstwertigen Bits verloren, sodass die weniger bedeutend Bits bleiben: 0, 1, 2 usw. Der Computer signalisiert das *Überlaufphänomen*. |
| 3.2 Organizarea și memorarea datelor |  | 3.2 Daten organisieren und speichern |
| Atât pentru reprezentarea întregilor, cât și pentru reprezentarea altor tipuri de date, orice sistem de calcul folosește o componentă specială, numită unitate de memorie. Prezentăm principalele elemente de structurare a acesteia.  Unitatea elementară de informație este bitul. Într-un bit se poate reprezenta o informație care poate să aibă doar două valori posibile: 0 sau 1. Interpretarea acestor valori se realizează în funcție de context: un bit poate să însemne 0 sau 1, *true* sau *false*, bărbat sau femeie, bine sau rău, alb sau negru etc. Din punct de vedere tehnologic, un bit este materializat prin niveluri de tensiune în curent continuu: 0 volți poate însemna ‚0’, în timp ce o tensiune de +5V poate însemna ‚1’.  Definiție: Un octet (*byte*) este o succesiune de 8 biți, numerotați de la 0 la 7, ca în **Figura 14**. |  | Jedes Computersystem verwendet sowohl für die Ganzzahlen Darstellung als auch für die Darstellung anderer Datentypen eine spezielle Komponente, die als Speichereinheit bezeichnet wird. Wir präsentieren die Hauptelemente der Strukturierung.  Die elementare Informationseinheit ist das Bit. Ein Bit kann Informationen darstellen, die nur zwei mögliche Werte haben können: 0 oder 1. Die Interpretation dieser Werte erfolgt über den Kontext: Ein Bit kann 0 oder 1 bedeuten, richtig oder falsch, männlich oder weiblich, gut oder schlecht, weiß oder schwarz etc. Aus technologischer Sicht wird ein Bit durch Gleichspannungspegel materialisiert: 0 Volt kannst ’0’ bedeuten, während eine Spannung von +5 V ’0’ bedeuten kann.  Definition: Ein Byte (Byte) ist eine 8-Bit-Sequenz, die wie in Figura 14 von 0 bis 7 nummeriert ist. |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 7 | 6 | 4 | 5 | 3 | 2 | 1 | 0 | | Bit *high*  (MSB) |  |  |  |  |  |  | Bit *low*  (LSB) |   **Figura 14.** *Numerotarea biților în cadrul unui octet* (Nummerierung der Bits in einem Byte) | | |
| *Octetul este unitatea elementară de adresare a memoriei*. Fiecare octet are atașat un număr întreg și nenegativ numit *adresa* octetului respectiv. Primul octet din memorie are adresa 0, al doilea octet are adresa 1, al treilea are adresa 2 ș.a.m.d. Sistemul poate referi / identifica fiecare octet din memorie folosind adresa acestuia. Intuitiv, ne putem imagina memoria ca o mulțime de căsuțe (așa cum sunt cele de la post-restant). Căsuțele sunt numerotate începând de la 0 și în fiecare căsuță poate fi un singur număr. În momentul în care depunem în căsuță un alt număr, vechiul număr se pierde (ca și la înregistrările audio / video pe bandă, rămâne doar ultima înregistrare). Numărul de pe ușa căsuței reprezintă adresa / referința, iar numărul din căsuță reprezintă *conținutul*.  Referirea la un octet se face, deci, prin adresa lui. De multe ori se practică referirea la un octet nu prin adresa lui, ci prin poziția lui față de un alt octet. În primul caz vorbim de *adresa absolută* a unui octet, iar în al doilea caz vorbim de *adresa relativă* față de un alt octet. În al doilea caz adresa absolută se obține adunând la adresa octetului de referință adresa relativă. De exemplu, dacă un octet A are adresa absolută 5643 și un octet B are adresa relativă 5 față de A, atunci adresa absolută a octetului B este 5643 + 5 = 5648. Ca terminologie, spunem că octetul B este cu 5 octeți mai la dreapta decât A. Vezi în **Figura 15** ilustrarea acestei situații. |  | *Das Byte ist die grundlegende Speicheradressiereinheit.* Jedes Byte hat eine ganz und eine nicht negative Zahl, die als Adresse dieses Bytes bezeichnet wird. Das erste Byte des Speichers hat die Adresse 0, das zweite Byte hat die Adresse 1, das dritte Byte hat die Adresse 2 und so weiter. Das System kann jedes Byte des Speichers anhand seiner Adresse referenzieren / identifizieren. Intuitiv können wir uns das Speicher als viele Hütten vorstellen (wie sie aus der Nachruhe stammen). Die Kästchen sind beginnend mit 0 nummeriert und jedes Kästchen kann eine einzelne Nummer sein. Wenn wir eine andere Nummer in die Box legen, geht die alte Nummer verloren (wie bei Audio- / Videobändern bleibt es nur die letzte Aufzeichnung). Die Nummer an der Tür der Box ist die Adresse / Referenz, und die Nummer in der Box ist der *Inhalt*.  Der Verweis auf ein Byte erfolgt durch seinem Adress. Es wird oft verwendet, um auf ein Byte nicht durch seine Adresse, sondern durch seine Position zu einem anderen Byte zu verweisen. Im ersten Fall beziehen wir uns auf die *absolute Adresse* eines Bytes und im zweiten Fall in Bezug auf die *relative Adresse* eines anderen Bytes. Im zweiten Fall wird die absolute Adresse erhalten, indem die relative Adresse zur Referenzbyteadresse addiert wird. Wenn zum Beispiel ein Byte A eine absolute Adresse von 5643 hat und ein Byte B eine relative Adresse von 5 in Bezug auf A hat, dann ist die absolute Adresse von Byte B 5643 + 5 = 5648. Als Terminologie sagen wir, dass Byte B 5 Bytes rechts ist als A. Siehe Figura 15 zur Veranschaulichung dieser Situation. |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | Conținut (*Inhalt*) octet 0  Adresa  0 | Conținut  (*Inhalt*) octet 1  Adresa  1 | Conținut  (*Inhalt*) octet 2  Adresa  2 | … | Conținut  (*Inhalt*) octet 5643  Adresa  5643 | … | Conținut  (*Inhalt*) octet 5648  Adresa  5648 | … |   **Figura 15.** *Succesiunea octeților în memorie* (Reihenfolge der Bytes im Speicher) | | |
| Bitul 0 al octetului se numește bitul cel mai puțin semnificativ (LSB), bitul de rang minim, bitul cel mai din dreapta, bitul *low* etc. Bitul 7 este bitul cel mai semnificativ (MSB), bitul de rang maxim, bitul cel mai din stânga, bitul *high* etc.  Să clarificăm două noțiuni fundamentale: **adresă – adresare** și **conținut – numerotare conținut**. În ceea ce privește adresarea: este unanim acceptată convenția că adresele octeților în memorie cresc de la stânga la dreapta (**Figura 15**); deci la adresare relativă, octetul de referință este la stânga octetului curent. În ceea ce privește conținutul unei locații de memorie: numerotările entităților dintr-un conținut se fac de la dreapta spre stânga, așa cum am văzut la numerotarea biților unui octet (**Figura 14**).  Entitatea octet este folosită practic de către toate instrucțiunile de prelucrare și de schimb cu exteriorul ale unui sistem de calcul. În contextul comunicațiilor trebuie reținut postulatul că: octetul are aceeași reprezentare, indiferent de sistemul de calcul. Cu alte cuvinte, se spune că octetul este portabil.  Accesul la biții unui octet se poate face prin intermediul unor instrucțiuni specializate. În particular, sunt situații în care se folosește ca entitate de execuție semioctetul (*nibble*). Un semioctet este format din patru biți, alăturați. Vorbim de semioctet low, sau semioctet mai puțin semnificativ, sau semioctet drept, sau cifră hexazecimală dreaptă, sau *nibble* drept etc. Analog, vorbim de semioctet high, sau semioctet semnificativ, sau semioctet stâng, sau cifră hexazecimală stângă, sau *nibble* stâng etc. Schematic, această împărțire apare ca în **Figura 16**. |  | Das Bit 0 des Bytes wird das niedrigstwertige Bit (LSB), das niedrigste Bit, das Bit ganz rechts, das *low* Bit usw. genannt. Bit 7 ist das höchstwertige Bit (MSB), das Bit mit dem höchsten Rang, das Bit ganz links, das *high* Bit usw.  Lassen Sie uns zwei grundlegende Begriffe klarstellen: **Adresse - Adressierung** und **Inhalt - Nummerierung von Inhalt**. In Bezug auf die Adressierung gilt allgemein die Konvention, dass die Adressen der Bytes im Speicher von links nach rechts zunehmen (Figura 15). Im Falle von relativen Adressierung befindet sich das Referenzbyte links vom aktuellen Byte. Zum Inhalt eines Speicherorts: Die Nummerierung der Entitäten in einem Inhalt erfolgt von rechts nach links, wie wir an der Nummerierung der Bits eines Bytes gesehen haben (Figura 14).  Die Byte-Entität wird praktisch von allen Verarbeitungs- und Austauschanleitungen mit der Außenseite eines Computersystems verwendet. Im Zusammenhang mit der Kommunikation ist zu beachten, dass das Byte unabhängig vom Rechensystem die gleiche Darstellung hat. Mit anderen Worten, das Byte soll portabel sein.  Der Zugriff auf die Bits eines Bytes kann durch spezielle Anweisungen erfolgen. Insbesondere gibt es Situationen, in denen das Halbbyte (*Nibble*) als Ausführungentität verwendet wird. Ein Halbbyte besteht aus vier Bits, die zusammengefügt werden. Wir sprechen von Low Halbbyte, oder niedrigstwertige Halbbyte, oder rechter Halbbyte, oder rechter Hexadezimale Ziffer, oder rechter *Nibble* etc. In analoger Weise sprechen wir von High Halbbyte, oder höchstwertige Halbbyte, oder linker Halbbyte, oder linker hexadezimaler Ziffer, oder linker *Nibble* usw. Schematisch erscheint diese Unterteilung wie in Abbildung 16. |
| |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 7 | 6 | 4 | 5 | 3 | 2 | 1 | 0 | |  |  |  |  |  |  |  |  | | Semioctetul high | | | | Semioctetul low | | | |   **Figura 16.** *Semiocteții componenți ai unui octet* (Halb-Byte-Komponenten eines Bytes) | | |
| Prelucrările fundamentale sunt efectuate pe octeți și pe grupuri de octeți consecutivi. În particular, operațiile cu numere întregi se pot efectua fie pe octeți, fie pe grupuri de octeți consecutivi. |  | Grundlegende Verarbeitung werden von Bytes und aufeinanderfolgenden Bytegruppen ausgeführt. Insbesondere können Ganzzahloperationen entweder byteweise oder am aufeinanderfolgende Bytegruppen ausgeführt werden. |
| O succesiune de octeți consecutivi de dimensiune fixată, privită ca o entitate de sine stătătoare formează o ***locație*** sau ***unitate de prelucrare***.  ***Adresa unei locații*** este egală cu adresa primului octet component al locației (cu cea mai mică adresă a octeților ce o compun).  ***Dimensiunea*** unei locații este egală cu numărul de octeți care o compun.  Dimensiunea și denumirea pe care o poartă o locație, variază de la un tip de sistem de calcul la altul. De exemplu, la unele sisteme, cum ar fi cele din familia IBM-PC:   * doi octeți consecutivi formează un *cuvânt*; * patru octeți consecutivi formează un *dublucuvânt*.   La astfel de sisteme vorbim de locații octet, locații cuvânt și locații dublucuvânt. În **Figura 17** prezentăm un cuvânt cu subdiviziunile (sub-entitățile) lui, iar în **Figura 18** prezentăm un dublucuvânt cu subdiviziunile (sub-entitățile) lui. |  | Eine Reihe aufeinanderfolgender Festgröβe-Bytes, die als eigenständige Entität betrachtet werden, bilden einen ***Speicherort*** oder eine ***Verarbeitungseinheit***.  ***Die Adresse eines Speicherorts*** entspricht der Adresse des ersten Bytes des Speicherorts (mit der kleinsten Byteadresse von sein Byten).  ***Die Größe*** eines Speicherorts entsprichts der Anzahl der von ihm Bytes.  Die Größe und der Name eines Speicherorts variieren von einem Typ von Computersystem zu einem anderen. Beispiel: Auf einigen Systemen, z. B. in der IBM-PC-Familie:   * zwei aufeinanderfolgende Bytes bilden einen *Wort*; * Vier aufeinanderfolgende Bytes bilden einen *Doppelwort*.   In solchen Systemen sprechen wir von Byte- Speicherorten, Wortspeicherorten und Doppeltwortspeicherorten. InFigura 17 präsentieren wir einen Wort mit seinen Unterteilungen (Unter-Entitäten), und in Figura 18 präsentieren wir ein Doppelwort mit seinen Unterteilungen (Unter-Entitäten). |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | Semioctet (*Halb-Byte*) 3 | | | | Semioctet (*Halb-Byte*) 2 | | | | Semioctet (*Halb-Byte*) 1 | | | | Semioctet (*Halb-Byte*) 0 | | | | | Octet 1 | | | | | | | | Octet 0 | | | | | | | |   **Figura 17.** *Un cuvânt IBM-PC* (Einen IBM-PC Wort) | | |
| Evident, pentru cuvântul din **Figura 17**, semioctetul 0 și octetul 0 sunt respectiv semioctetul low și octetul low din cadrul locației. Similar, semioctetul 3 și octetul 1 sunt respectiv semioctetul high și octetul high din cadrul locației. |  | Offensichtlich sind für das Wort in Figura 17 das Halb-Byte 0 und das Byte 0 das Low Halb-Byte und das Low Byte innerhalb den Speicherort sind. In ähnlicher Weise das Halb-Byte 3 und das Byte 1 das High Halb-Byte und das High Byte innerhalb den Speicherort sind. |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 31 | 30 | 29 | 28 | 27 | 26 | 2  5 | 2  4 | 2  3 | 2  2 | 2  1 | 2  0 | 1  9 | 1  8 | 1  7 | 1  6 | 1  5 | 1  4 | 1  3 | 1  2 | 1  1 | 1  0 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | Semioctet 3 | | | | Semioctet 2 | | | | Semioctet 1 | | | | Semioctet 0 | | | | Semioctet 3 | | | | Semioctet 2 | | | | Semioctet 1 | | | | Semioctet 0 | | | | | Octet (*Byte*) 3 | | | | | | | | Octet (*Byte*) 2 | | | | | | | | Octet (*Byte*) 1 | | | | | | | | Octet (*Byte*) 0 | | | | | | | | | Cuvânt (*Wort*) 1 | | | | | | | | | | | | | | | | Cuvânt (*Wort*) 0 | | | | | | | | | | | | | | | |   **Figura 18.** *Un dublucuvânt IBM-PC* (Einen IBM-PC Doppelwort) | | |
| Pentru dublucuvântul din **Figura 18**, semioctetul 0, octetul 0 și cuvântul 0 sunt respectiv semioctetul low, octetul low și cuvântul low din cadrul locației. Similar, semioctetul 7, octetul 3 și cuvântul 1 sunt respectiv semioctetul high, octetul high și cuvântul high din cadrul locației. |  | Für das Doppelwort in Figura 18 sind der Halb-Byte 0, das Byte 0 und das Wort 0 jeweils das Low (niedrigstwertige) Halb-Byte, das Low Byte und das Low Wort an der Speicherorts. Ebenso sind der Halb-Byte 7, das Byte 3 und das Wort 1 jeweils das High (höchstwertige) Halb-Byte, das High Byte und das High Wort an der Speicherorts. |
| 3.3 Tipuri elementare de date: dimensiuni ale standardelor de reprezentare |  | 3.3 Elementare Datentypen: Dimensionen von die Darstellungsstandarden |
| Cele mai utilizate tipuri elementare de date sunt caracterele, numerele întregi și numerele reale. Pentru fiecare dintre aceste tipuri de date sunt stabilite o serie de standarde de reprezentare a datei în memorie. În particular, standardul de reprezentare fixează dimensiunea celulei de memorie (*memory location*) pentru fiecare tip de dată.  Câteva dimensiuni ale locațiilor de reprezentare:   * Tip de date caracter. * 1 octet - standardul ASCII; * 2 octeți-standardul UNICODE. * Tip de date întreg: 1, 2, 4, 8 octeți, depinde de sistemul de calcul. * Tip de date real: * 4 octeți – standardul IEEE simplă precizie; * 8 octeți – standardul IEEE dublă precizie; * 6 octeți – standardul Turbo Pascal.   Câteva exemple de reprezentări ale unor date elementare. Conținuturile locațiilor le vom scrie în hexazecimal:   * *Caracterele* „M” și „m” se reprezintă: * 4D, respectiv 6D – standardul ASCII; * 004D, respectiv 006D – standardul UNICODE. * Numerele 1 și -1, interpretate ca întregi pe 2 octeți se reprezintă 0001 respectiv FFFF (deoarece se folosește sistemul de numerație Complementul lui 2). Aceleași numere interpretate ca întregi pe 4 octeți se reprezintă 00000001 respectiv FFFFFFFF; * Numerele 2 și -2, interpretate ca întregi pe 2 octeți se reprezintă 0002 respectiv FFFE. Aceleași numere interpretate ca întregi pe 4 octeți se reprezintă 00000002 respectiv FFFFFFFE. * Numerele 1 și -1, dar de această dată interpretate ca numere reale se reprezintă: * 3F800000, respectiv BF800000 – standardul IEEE simplă precizie, * 3FF0000000000000, respectiv BFF0000000000000 – standardul IEEE dublă precizie * 000000000081, respectiv 800000000081 – standardul Turbo Pascal.   Conform cu cele prezentate la numerotarea octeților într-o locație, pentru exemplele de mai sus, avem:   * Octetul 0 al reprezentării caracterului „m” în standardul UNICODE are valoarea 6D; * Octetul 1 al reprezentării caracterului „m” în standardul UNICODE are valoarea 00; * Octetul 0 al reprezentării numărului -1 în standardul Turbo Pascal are valoarea 81; * Octetul 5 al reprezentării numărului -1 în standardul Turbo Pascal are valoarea 80; * Octetul 0 al reprezentării numărului -1 în standardul IEEE dublă precizie are valoarea 00; * Octetul 6 al reprezentării numărului -1 în standardul IEEE dublă precizie are valoarea F0; * Octetul 7 al reprezentării numărului -1 în standardul IEEE dublă precizie are valoarea BF. |  | Die am häufigsten verwendeten elementaren Datentypen sind Zeichen, Ganzzahlen und Realer Zahlen. Für jeden dieser Datentypen ist eine Reihe von Standards für die Datendarstellung im Speicher festgelegt. Insbesondere legt der Darstellungsstandard die Größe des Speicherplatzes (*memory location*) für jeden Datentyp fest.  Verschiedene Dimensionen von Darstellungsorten:   * Zeichen Datentyp. * 1 Byte – ASCII Standard; * 2-Byten – UNICODE Standard. * Ganzzahlen Datentyp: 1, 2, 4, 8 Byten, abhängig vom Computersystem. * Realer Datentyp: * 4 Byten – IEEE *einfache Präzision* Standard; * 8 Bytes – IEEE *doppelte Präzision* Standard; * 6 Bytes – der Turbo Pascal Standard.   Einige Beispiele für die Darstellung von grundlegende Daten. Wir werden den Inhalt den Speciherorten in hexadezimal schreiben:   * Die Zeichen ”M” und ”m” sind: * 4D oder 6D – der ASCII-Standard; * 004D und 006D – der UNICODE-Standard. * Zahlen 1 und -1, interpretiert als 2-Byten-Ganzzahlen, werden durch 0001 bzw. FFFF dargestellt (da das Nummerierungssystem von Zweier Complement verwendet wird). Dieselben Zahlen, die als 4-Byten-Ganzzahlen interpretiert werden, sind 00000001 bzw. FFFFFFFF; * Zahlen 2 und -2, die als 2-Byten-Ganzzahlen interpretiert werden, werden durch 0002 bzw. FFFE dargestellt. Dieselben Zahlen, die als 4-Byten-Ganzzahl interpretiert werden, sind 00000002 bzw. FFFFFFFE. * Die Zahlen 1 und -1, aber diesmal jedoch als reelle Zahlen interpretiert, sind: * 3F800000 bzw. BF800000 – in der einfache Präzision IEEE Standard; * 3FF0000000000000 bzw. BFF0000000000000 – in der doppelte Präzision IEEE-Standard; * 000000000081 bzw. 800000000081 – in der Turbo Pascal Standard.   Entsprechend den in der Nummerierung der Bytes an einer Speicherort gezeigten haben wir für die obigen Beispiele:   * Das Byte 0 der Zeichendarstellung ”m” im UNICODE-Standard ist 6D; * Das Byte 1 der Zeichendarstellung "m" im UNICODE-Standard ist 00; * Das Byte 0 der -1-Darstellung im Turbo Pascal-Standard ist 81; * Das Byte 5 der -1-Darstellung im Turbo Pascal-Standard ist 80; * Das Byte 0 der -1-Darstellung im IEEE-Standard mit doppelte Präzision ist 00; * Das Byte Stelle der -1-Darstellung im IEEE-Standard mit doppelte Präzision ist F0; * Das Byte 7 der -1-Darstellung im IEEE-Standard mit doppelter Präzision ist BF. |
| 3.4 Ordinea octeților într-o locație; mașini little-endian și mașini big-endian |  | 3.4 Die Reihenfolge der Bytes an einem Speicherort; little-endian und big-endian Maschinen |
| La nivelul unei locații, privită ca o entitate de sine stătătoare cu conținut interpretat în funcție de standardul de reprezentare, se disting două abstracțiuni:   * Numerotarea octeților în cadrul unei locații fixată de standardul de reprezentare; * Adresele octeților care compun locația.   Așa cum am precizat deja, numerotările de conținuturi se fac de la dreapta spre stânga, iar adresele cresc de la stânga spre dreapta. În mod normal, se pune problema unei corespondențe între aceste două elemente.  Pe de o parte vorbim de *reprezentarea structurală* a unui tip de date, impusă de standardul de reprezentare. Pe de altă parte vorbim de *memorarea concretă* a datei în locație: în care octet al locației memorăm octetul 0 al reprezentării, în care octet al locației memorăm octetul 1 ș.a.m.d.  Cu alte cuvinte trebuie stabilită o *corespondență* între ordinea octeților impusă de reprezentarea structurală și adresele din locație în care se memorează valorile acestor octeți.  Această corespondență constituie o caracteristică a sistemului de calcul și ea poate fi una dintre următoarele două:   * Plasarea *little-endian*,în care octetul cu cea mai mică adresă din locație va conține octetul cu numărul 0 al reprezentării, octetul cu adresa următoare va conține octetul 1 al reprezentării ș.a.m.d. (octetul „end” al reprezentării are adresa cea mai mică, „little”); * Plasarea *big-endian*, în care octetul cu cea mai mare adresă din locație va conține octetul 0 al reprezentării, octetul cu adresa precedentă va conține octetul 1 al reprezentării ș.a.m.d. (octetul „end” al reprezentării are adresa cea mai mare „big”).   Spre exemplu, urmărim să reprezentăm numărul (1025)10 într-o locație de patru octeți. Pentru această dimensiune a locației, reprezentările lui în bazele 16 și 2 sunt (00000401)16, respectiv (00000000 00000000 00000100 00000001)2. Să presupunem că **B** este adresa locației în care numărul este plasat în ordinea big-endian, iar **L** este adresa locației în care același număr este plasat în ordinea little-endian. Cele două plasări sunt ilustrate în **Figura 19**, cu conținuturile octeților scrise atât în hexazecimal, cât și în binar: |  | Auf der Ebene eines Speicherorts, der als eigenständige Entität mit interpretiertem Inhalt gemäß dem Repräsentationsstandard betrachtet wird, werden zwei Abstraktionen unterschieden:   * Nummerierung der Bytes in einer durch den Darstellungsstandard festgelegten Speicherort; * Adressen von Bytes, die der Speicherort bilden.   Wie bereits erwähnt, werden die Inhalte von rechts nach links nummeriert und die Adressen von links nach rechts erhöht. Normalerweise besteht eine Frage der Übereinstimmung zwischen diesen beiden Elementen.  Einerseits geht es um die *strukturelle Repräsentation* eines Datentyps, der durch den Repräsentationsstandard vorgegeben ist. Andererseits geht es um die *tatsächliche Speicherung* des Datums am Speicherort: in welchem Byte des Speicherorts speichern wir das Byte 0 der Darstellung, in welchem Byte des Speicherorts speichern wir Byte 1 usw.  Mit anderen Worten eine *Korrelation* zwischen der Reihenfolge des Byten, die die Strukturdarstellung auferlegten und die Adressen an dem Speicherort, an dem die Werte dieser Bytes gespeichert werden sollen.  Diese Entsprechung ist ein Merkmal des Computersystems und kann eine der folgenden zwei sein:   * *Little-Endian*-Platzierung: Das Byte mit der kleinsten Adresse am Speicherort enthält das 0-Byte der Darstellung, das Byte mit der nächsten Adresse enthält das Byte 1 der Darstellung usw. (Das „End“ -Byte der Darstellung hat die kleinste Adresse, „*little*“);   • *Big-Endian*-Platzierung, bei der das Byte mit der größten Adresse am Speicherort enthält das 0-Byte der Darstellung, das Byte mit der vorherigen Adresse enthält das Byte 1 der Darstellung usw. (Das „End“ -Byte der Darstellung hat die größte Adresse, „*big*“).  Zum Beispiel versuchen wir, die Zahl (1025)10 an einer 4-Byte-Stelle darzustellen. Für diese Dimension des Speicherorts sind seine Darstellungen in den Basen 16 und 2 (00000401)16 bzw. (00000000 00000000 00000100 00000001)2. Angenommen, **B** ist die Adresse des Speicherortes, an dem die Zahl in der Big-Endian-Reihenfolge platziert ist, und **L** ist die Adresse des Speicherortes, an dem die gleiche Zahl in der Little-Endian-Reihenfolge platziert ist. Die beiden Platzierungen sind in Figura 19 mit hexadezimalem und binärem Byte-Inhalt dargestellt: |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | *Big-endian* | 00 | 00 | 04 | 01 | | 00000000 | 00000000 | 00000100 | 00000001 | |  | Adresa (*Adresse*) | Adresa (*Adresse*) | Adresa (*Adresse*) | Adresa (*Adresse*) | |  | B | B+1 | B+2 | B+3 |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | *Little-endian* | 01 | 04 | 00 | 00 | | 00000001 | 00000100 | 00000000 | 00000000 | |  | Adresa (*Adresse*) | Adresa (*Adresse*) | Adresa (*Adresse*) | Adresa (*Adresse*) | |  | L | L+1 | L+2 | L+3 |   **Figura 19.** *Plasările big-endian și little-endian* (Die big-endian und little-endian Platzierungen) | | |
| Plasarea little-endian sau big-endian este o caracteristică a sistemului de calcul. De multe ori se folosește termenul de *arhitectură big-endian*, respectiv *arhitectură little-endian*. Un procesor are instrucțiuni specializate care să opereze cu fiecare tip de locație, instrucțiuni care „știu” standardul de reprezentare și ordinea de plasare. Utilizatorul trebuie doar să comande procesorului operația dorită și adresa locației de unde aceasta să își ia reprezentarea datelor.  Fiecare dintre cele două moduri de plasare are avantaje și dezavantaje. Prezentăm doar două criterii de comparare, fiecare dintre ele fiind avantajos pentru o arhitectură și dezavantajos pentru cealaltă. Aceste criterii sunt:   * Conversia unui număr întreg de la o reprezentare mai mare la una mai mică. De exemplu, dacă se cere ca un întreg dintr-o locație de 4 octeți, dar care încape de fapt pe 2 octeți, să fie prelucrat, cu aceeași valoare, ca și când ar face parte dintr-o locație pe 2 octeți. Comparativ, acest criteriu reprezintă un avantaj little-endian, dezavantaj big-endian. * Depistarea biților de semn. Este unanim acceptat faptul că semnul unui număr, întreg sau real, se reprezintă pe un bit, cu valoarea 0 pentru număr pozitiv și cu valoarea 1 pentru număr negativ. Indiferent de standardul de reprezentare, bitul de semn este bitul high al octetului high din reprezentare. Comparativ, acest criteriu reprezintă un avantaj big-endian, dezavantaj little-endian.   De ce? Să luăm ca exemplu reprezentările numărului 2 pe 4 octeți, atât în plasarea big-endian, cât și în plasarea little-endian, ilustrate în **Figura 20**.  Pentru conversie, în cazul unei mașini little endian adresa locației rămâne L, indiferent dacă aceasta are dimensiunea de 4 octeți, de 2 octeți, sau chiar de 1 octet: se modifică doar dimensiunea locației (deci locația care conține numărul 2 va avea aici aceeași adresă L, indiferent dacă numărul este reprezentat pe 1, 2 sau 4 octeți!). La mașina big-endian, adresele locațiilor trebuie modificate în funcție de dimensiunea acestora: B+2 pentru locația de 2 octeți (conținând octeții de adrese B+2 și B+3) și B+3 pentru locația de 1 octet. Deci în cazul big-endian procesorul trebuie să facă în plus calculul de adresă. |  | Little-Endian- oder Big-Endian-Platzierung ist ein Merkmal des Computersystems. Oft wird der Begriff „*Big-Endian-Architektur*” oder „*Little-Endian-Architektur*” verwendet. Ein Prozessor hat spezielle Anleitungen, um mit jeder Art von Speicherort zu arbeiten, Anleitungen, die den Repräsentationsstandard und die Reihenfolge der Platzierung ”kennen”. Der Benutzer muss dem Prozessor nur die gewünschte Operation und die Adresse des Speicherortes befehlen, von dem er seine Datendarstellung bezieht.  Jede der beiden Platzierungen hat Vor- und Nachteile. Wir stellen nur zwei Vergleichskriterien vor, von denen jedes für eine Architektur vorteilhaft und das andere nachteilig ist. Diese Kriterien sind:   * Das Konvertieren einer Ganzzahl von einer größeren in eine kleinere Darstellung. Wenn beispielsweise eine 4-Byte-Speicherort insgesamt, die jedoch tatsächlich 2 Byte umfasst, mit demselben Wert verarbeitet werden muss, als wäre sie Teil einer 2-Byte-Position. Im Vergleich ist dieses Kriterium ein vorteilhaft für Little-Endian, aber ein Nachteil für Big-Endian. * Vorzeichenbit erkennen. Es ist allgemein anerkannt, dass das Vorzeichen einer ganzen oder reellen Zahl durch ein Bit mit dem Wert 0 für die positive Zahl und dem Wert 1 für die negative Zahl dargestellt wird. Unabhängig vom Darstellungsstandard ist das Vorzeichenbit das High-Bit der High-Byte-Darstellung. Im Vergleich ist dieses Kriterium ein vorteilhaft für Big-Endian, aber ein Nachteil für Little-Endian.   Warum? Nehmen wir als Beispiel die Darstellungen von der Zahl 2 um 4 Byten sowohl in der Big-Endian-Platzierung als auch in der Little-Endian-Platzierung (siehe Figura 20).  Für der Konvertierung bei einem Little-Endian-Rechner bleibt die Speicherortadresse L, unabhängig davon, ob sie 4 Byte, 2 Byte oder sogar 1 Byte groß ist: Nur die Größe des Speicherorts ändert sich (der Speicherort mit die Zahl 2 hat hier also die gleiche Adresse L, unabhängig davon, ob die Zahl durch 1, 2 oder 4 Bytes dargestellt wird!). Für den Big-Endian-Rechner müssen die Standortadressen entsprechend ihrer Größe geändert werden: B + 2 für den 2-Byte-Standort (mit B + 2- und B + 3-Adressbytes) und B + 3 für den 1-Byte- Speicherort. Im Big-Endian-Fall muss der Prozessor die Adressberechnung also weiter ausführen. |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | *Big-endian* | 00 | 00 | 00 | 02 | | 00000000 | 00000000 | 00000000 | 00000010 | |  | Adresa (*Adresse*) | Adresa (*Adresse*) | Adresa (*Adresse*) | Adresa (*Adresse*) | |  | B | B+1 | B+2 | B+3 |  |  |  |  |  |  | | --- | --- | --- | --- | --- | | *Little-endian* | 02 | 00 | 00 | 00 | | 00000010 | 00000000 | 00000000 | 00000000 | |  | Adresa (*Adresse*) | Adresa (*Adresse*) | Adresa (*Adresse*) | Adresa (*Adresse*) | |  | L | L+1 | L+2 | L+3 |   **Figura 20.** *Reprezentarea lui 2 în cele două tipuri de plasări* (Die Darstellung von 2 in den zwei Arten von Platzierungen) | | |
| Pentru bitul de semn, în cazul unei mașini big-endian adresa octetului cu bitul de semn coincide cu adresa locației. Cu notațiile din **Figura 20**, bitul de semn, la mașina big-endian, se află la adresa B, indiferent de faptul că se va prelucra o locație de 4 octeți, de 2 octeți sau de 1 octet. În cazul unei mașini little-endian, bitul de semn se află în ultimul octet al locației (cel cu cea mai mare adresă), așa că pentru a-1 obține procesorul trebuie să calculeze adresa acestui octet: ea este L pentru locația de 1 octet, L+1 pentru locația de 2 octeți și L+3 pentru cea de 4 octeți.  Utilizatorul poate să interpreteze în mod diferit conținutul aceleiași locații! De exemplu, poate să memoreze într-un șir de 4 octeți un număr întreg, după care să comande operarea asupra aceleiași arii de memorie prin patru instrucțiuni care utilizează 4 locații consecutive de tip caracter reprezentat pe octet. În astfel de situații, când la momente de timp diferite se interpretează diferit aceeași arie de memorie, trebuie să se țină cont de ordinea de plasare și de standardele de reprezentare ale datelor elementare.  Sistemele de calcul de dimensiuni mari, cum ar fi procesoarele SPARC sau MOTOROLA, mașinile RISC, ca și supercalculatoarele CDC-Cyber sau CRAY, folosesc arhitectura big-endian. Calculatoarele uzuale actuale, în particular cele de tip IBM-PC, procesoarele INTEL și DEC-Alpha folosesc arhitectură little-endian. De o factură aparte sunt calculatoarele din familia PowerPC, care sunt mașini big-endian, ele „înțelegând” ambele arhitecturi. |  | Für das Vorzeichenbit stimmt im Fall einer Big-Endian-Maschine die Adresse des Bytes mit dem Vorzeichenbit mit der Adresse des Speicherorts überein. Mit den Notationen in Figura 20 befindet sich das Vorzeichenbit auf der Big-Endian-Maschine auf B, unabhängig davon, ob eine 4-Byte-, 2-Byte- oder 1-Byte- Speicherort verarbeitet wird. Bei einer Little-Endian-Maschine befindet sich das Vorzeichenbit im letzten Byte der Speicherort (dem mit der größten Adresse). Um den Prozessor zu erhalten, muss er die Adresse dieses Bytes berechnen: Es ist L für die 1-Byte-Speicherort, L + 1 für die 2-Byte-Speicherort und L + 3 für die 4 Bytes-Speicherort.  Der Nutzer kann den Inhalt desselben Ortes unterschiedlich interpretieren! Beispielsweise kann es eine Ganzzahl in einer Folge von 4 Bytes speichern und dann denselben Speicherbereich mit vier Anleitungen befehlen, die vier aufeinanderfolgende Zeichenspeichernorten verwenden, die durch Byte dargestellt werden. In solchen Situationen wenn zu verschiedenen Zeiten derselbe Speicherbereich unterschiedlich interpretiert wird, müssen die Platzierungsreihenfolge und die Darstellungstandard für die elementaren Daten berücksichtigt werden.  Großrechnersysteme wie SPARC- oder MOTOROLA-Prozessoren, RISC-Maschinen und CDC-Cyber- oder CRAY-Supercomputer verwenden eine Big-Endian-Architektur. Gegenwärtige Computer, insbesondere IBM-PC-, INTEL- und DEC-Alpha-Prozessoren, verwenden eine Little-Endian-Architektur. Einem besonderen Typen sind die PowerPC-Computer, bei denen es sich um Big-Endian-Maschinen handelt, die beide Architekturen „verstehen“. |
| 3.5 Unități de capacitate a memoriei |  | 3.5 Speicherkapazitätseinheiten |
| Prin capacitatea de memorare a unui sistem de calcul înțelegem numărul total de octeți ai unității de memorie.  În practică se folosesc o serie de multipli ai numărului de octeți. Spre deosebire de multiplii folosiți în activitatea cotidiană, unitățile multipli ale capacității de memorare sunt exprimate sub formă de puteri ale lui 2, astfel:  1 Ko Kilo-octet = 210 octeți  1 Mo Mega-octet = 220 octeți  1 Go Giga-octet = 230 octeți  1 To Tera-octet = 240 octeți. |  | Unter der Speicherkapazität eines Computersystems versteht man die Gesamtzahl der Bytes der Speichereinheit.  In der Praxis wird eine Vielzahl von Bytes verwendet. Im Gegensatz zu den im Alltag verwendeten Vielfachen werden Mehrfachspeicherkapazitätseinheiten wie folgt als Zweierpotenzen ausgedrückt:  1 KB Kilo-Byte = 210 Byten  1 MB Mega-Byte = 220 Byten  1 GB Giga-Byte = 230 Byten  1 TB Terra-Byte = 240 Byten. |
| 3.6 Codificarea caracterelor |  | 3.6 Zeichenkodierung |
| Pentru codificarea caracterelor tipăribile în vederea prelucrării lor automate, s-au definit o serie standarde de reprezentare. Aceste standarde atribuie câte un număr întreg fiecărui caracter, iar valorile de codificare a caracterelor dintr-o anumită grupă respectă anumite condiții. Existența acestor condiții este benefică pentru prelucrarea automată a caracterelor.  Un prim sistem de codificare stabilit a fost EBCDIC (*Extended Binary Decimal Interchange Code*), care codifică un caracter pe un octet folosind numere întregi din intervalul [0,255]. În prezent, cel mai folosit sistem de codificare este ASCII (*American Standard Code for Information Interchange*). Standardul ASCII este un cod pe 7 biți, folosind numerele întregi din intervalul [0,127]. În ASCII este codificat un caracter pe un octet, iar bitul 7 (cel de-al 8-lea) este automat 0. Practic, toate calculatoarele actuale folosesc ASCII pentru codificarea caracterelor. Firma IBM a propus (și la propunere au aderat practic toate marile case de software și hardware), extinderea ASCII folosind și cel de-al 8-lea bit din octet, pentru codificarea unor caractere grafice speciale.  În contextul cerințelor de internaționalizare impuse de existența Internet, în ultimii 10 ani se impune din ce în ce mai mult standardul de codificare UNICODE. Acesta codifică un caracter pe doi octeți, pentru a se permite codificarea simbolurilor și a caracterelor folosite în scrierile majorității limbilor de pe planetă.  În cele ce urmează ne rezumăm la prezentarea standardului de codificare ASCII. Standardul ASCII împarte caracterele în următoarele cinci grupe:   1. Literele mici ale alfabetului a, b,..., z. 2. Literele mari ale alfabetului A, B,..., Z. 3. Cifrele zecimale 0, 1,... , 9. 4. O serie de caractere speciale: spațiul, virgula, punctul, +, -, \_, $, & ș.a.m.d 5. Set de caractere funcționale care nu apar la tipărire / afișare, ci doar dirijează tipărirea / afișarea.   Caracterele funcționale apar în tabelele de definiție sub forma unor grupuri de litere, ca de exemplu CR, LF, TAB, FF, BEL, BS ș.a.m.d. CR provoacă deplasarea dispozitivului de afișare (tipărire) la început de rând (*Carriage Return*). LF sau NL provoacă deplasarea dispozitivului cu un rând mai jos (*Line Feed* sau *New Line*), păstrându-se poziția în cadrul rândului. În funcție de sistem, pentru a separa două linii dintr-un text se folosește fie LF, fie succesiunea de caractere CR LF. TAB este caracterul de tabulare, deci provoacă avansul dispozitivului la poziția următorului stop de tabulare (de obicei peste 5-8 caractere). FF (*Form Feed*) provoacă trecerea la pagina (ecranul) următoare (următor). BEL provoacă emiterea unui semnal sonor, iar BS (*backspace*) provoacă deplasarea dispozitivului de afișare (tipărire) cu o poziție spre stânga, în vederea ștergerii (supraimprimării) ultimului caracter.  Condițiile de codificare pe care le respectă standardul ASCII sunt:   * Toate caracterele funcționale au codul mai mic decât codul caracterului spațiu; * Codul caracterului spațiu este mai mic decât codurile celorlalte caractere tipăribile; * Literele mici sunt codificate prin 26 numere consecutive, în ordine alfabetică; * Literele mari sunt codificate prin 26 numere consecutive, în ordine alfabetică; * Cifrele zecimale sunt codificate prin 10 numere consecutive, în ordinea valorilor. |  | Um die druckbaren Zeichen für ihre automatische Verarbeitung zu codieren, wurden eine Reihe von Darstellungsstandarden definiert. Diese Standarden weisen jedem Zeichen eine Ganzzahl zu, und die Zeichencodierungswerte in einer bestimmten Gruppe erfüllen bestimmte Bedingungen. Das Vorhandensein dieser Bedingungen ist für die automatische Zeichenverarbeitung von Vorteil.  Ein erstes festgelegtes Codierungssystem war EBCDIC (*Extended Binary Decimal Interchange Code*), das ein Zeichen in einem Byte mit Ganzzahlen im Bereich [0,255] codiert. Derzeit das gebräuchlichste Codierungssystem ist der *American Standard Code for Information Interchange* (ASCII). Der ASCII-Standard ist ein 7-Bit-Code, der Ganzzahlen im Bereich [0,127] verwendet. In ASCII wird ein Zeichen in einem Byte codiert, und Bit 7 (das achte) ist automatisch 0. Nahezu alle aktuellen Computer verwenden ASCII zum Codieren von Zeichen. IBM hat vorgeschlagen (und praktisch alle großen Software- und Hardwarehäuser haben sich dem Vorschlag angeschlossen), ASCII mithilfe des achte Bit von das Byte zu erweitern, um spezielle Grafikzeichen zu codieren.  Im Kontext der Internationalisierungsanforderungen, die das Internet gestellt hat, in den letzten 10 Jahren muss der UNICODE codieren-Standard immer mehr werden. Es codiert ein Zeichen durch zwei Bytes, um die Symbole und Zeichen zu codieren, die zum Schreiben der meisten Sprachen auf dem Planeten verwendet werden.  Im Folgenden fassen wir die Darstellung des ASCII-Kodierungsstandards zu präsentieren. Der ASCII-Standard unterteilt die Zeichen in die folgenden fünf Gruppen:   1. Die Kleinbuchstaben des Alphabets a, b, ..., z. 2. Großbuchstaben des Alphabets A, B, ..., Z. 3. Dezimalzahlen 0, 1, ..., 9. 4. Eine Reihe von Sonderzeichen: Leerzeichen, Komma, Punkt, +, -, \_, $, & etc. 5. Satz von Funktionszeichen, die nicht auf dem Druck / Display erscheinen, sondern nur direkt auf dem Druck / Display.   Funktionszeichen werden in den Definitionstabellen als Buchstabengruppen angezeigt, z. B. CR, LF, TAB, FF, BEL, BS usw. CR bewirkt, dass sich das Anzeigegerät (Print) an den oberen Rand der Zeile bewegt (*Carriage Return*). LF oder NL bewirkt, dass sich das Gerät eine Zeile nach unten bewegt [Zeilenvorschub (*Line Feed*) oder Neue Zeile (*New Line*)], wobei die Position in der Zeile beibehalten wird. Abhängig vom System wird zum Trennen von zwei Textzeilen entweder LF oder die CR LF-Zeichenfolge verwendet. TAB ist das Tabulatorzeichen, sodass das Gerät zur nächsten Tabulatorposition vorrückt (normalerweise mehr als 5 bis 8 Zeichen). Mit FF (*Form Feed*) wechseln Sie zur nächsten Seite oder Bildschirm. BEL gibt einen Piepton aus und BS [Rücktaste (*backspace*)] bewirkt, dass das Anzeigegerät (Drucken) eine Position nach links verschoben wird, um das letzte Zeichen zu löschen (überdrucken).  Die Kodierungsbedingungen, denen der ASCII-Standard entspricht, sind:   * Alle Funktionszeichen haben einen niedrigeren Code als der Leerzeichencode; * Der Leerzeichencode ist kleiner als die anderen Zeichen des druckbaren Zeichens; * Kleinbuchstaben werden durch 26 aufeinanderfolgende Zahlen in alphabetischer Reihenfolge codiert; * Große Buchstaben werden durch 26 aufeinander folgende Zahlen in alphabetischer Reihenfolge codiert; * Dezimalstellen werden durch 10 aufeinanderfolgende Zahlen in der Reihenfolge der Werte codiert. |
| F:\didactic\ASC\Documentatii\asciifull.gif  **Figura 21.** *Codul ASCII* (Die ASCII Code) | | |
| 3.7 Înmulțiri și împărțiri |  | 3.7 Multiplikationen und Divisionen |
| Înmulțirea și împărțirea pe *n* biți impun următoarele restricții de dimensiune a locațiilor:  Înmulțirea pe *n* biți presupune că ambii factori sunt reprezentați pe câte *n* biți, iar produsul lor va fi reprezentat pe 2 × *n* biți. În cazul convenției cu semn, factorii înmulțirii vor fi reprezentați în cod complementar. Produsul va fi reprezentat tot în cod complementar și va respecta regula semnelor.  Drept consecință imediată a acestei dimensionări, rezultă că operația de înmulțire nu provoacă depășire! Într-adevăr, în convenția fără semn cea mai mare valoare posibilă pe *n* biți este 2*n*-l, pătratul acestei valori este 22*n* - 2*n*+1 + l: acest număr încape pe 2*n* biți. În convenția cu semn, numărul -2*n*-1 are valoarea absolută cea mai mare, iar pătratul acesteia este 22*n*-2, număr care se poate reprezenta pe 2*n*-1 biți, deci în cod complementar acest număr se poate reprezenta pe 2*n* biți.  Împărțirea pe *n* biți (oarecum invers față de înmulțire), impune condiția ca deîmpărțitul să fie reprezentat pe 2 × *n* biți, iar împărțitorul pe *n* biți. Operația furnizează două rezultate: câtul reprezentat pe *n* biți și restul reprezentat tot pe *n* biți. În cazul convenției cu semn, deîmpărțitul și împărțitorul se vor reprezenta în cod complementar. Atât câtul, cât și restul vor fi, de asemenea, reprezentate în cod complementar. Câtul împărțirii va respecta regula semnelor. Important! Restul împărțirii va fi, în valoare absolută, mai mic decât valoarea absolută a împărțitorului și va avea același semn ca deîmpărțitul! De exemplu, -7 : 3 dă câtul -2 și restul -l, adică -7 = (-2) × 3 + (-1).  În comparație, teorema împărțirii cu rest din aritmetică spune că restul trebuie să fie un număr pozitiv, deci în aritmetică avem (-7) = (-3) × 3 + 2, deci câtul este -3 și restul 2!  Operația de împărțire semnalează eroare la împărțitor zero (*Divide by zero*!). În cazul neîncadrării în dimensiuni semnalează depășire! Spre exemplu, împărțirea fără semn 1000 : 3 se poate, formal, efectua, deoarece deîmpărțitul se poate reprezenta pe 16 biți și împărțitorul se poate reprezenta pe 8 biți. La această operație va apărea depășire, deoarece câtul este 333 și nu se poate reprezenta pe 8 biți! Restul împărțirii este 1 și se poate reprezenta pe un octet. Pentru a evita o astfel de situație, programatorul poate decide să facă operația pe 16 biți, adică să reprezinte deîmpărțitul pe 32 de biți și împărțitorul pe 16 biți, urmând să obțină câtul și restul pe câte 16 biți (333 încape pe 16 biți și astfel nu vom mai avea depășire).  *Operațiile de înmulțire și împărțire fără semn*  Înmulțireafără semn a numerelor întregi reprezentate binar se desfășoară conform algoritmului cunoscut, doar că se folosește baza 2. Pentru comoditate, vom considera *n* = 4. Să considerăm factorii 11 și 13. Înmulțirea lor în baza 2 este: |  | Durch Multiplikation und Division in *n* Bits werden die folgenden Einschränkungen der Speicherorten auferlegt:  Die Multiplikation auf *n* Bits impliziert, dass beide Faktoren durch *n* Bits dargestellt werden und ihr Produkt durch 2 × *n* Bits dargestellt wird. Im Falle einer Vorzeichenkonvention werden Multiplikationsfaktoren in einem komplementären Code dargestellt. Das Produkt wird auch im komplementären Code dargestellt und folgt der Vorzeichenregel.  Als unmittelbare Folge dieser Dimensionierung ergibt sich, dass die Multiplikation kein Überschwingen verursacht! Wirklich ist in der vorzeichenlosen Konvention der höchstmögliche Wert für *n* Bits 2*n*-1, das Quadrat dieses Werts ist 22*n* - 2*n* + 1 + 1: diese Zahl passt auf 2*n* Bits. In der Vorzeichenkonvention hat die -2*n*-1-Zahl den höchsten Absolutwert und ihr Quadrat ist 22*n*-2, die durch 2*n*-1-Bits dargestellt werden kann. In einem komplementären Code kann diese Zahl also durch 2*n*-Bits dargestellt werden.  Die Division auf *n* Bits (etwas umgekehrt zur Multiplikation) impliziert, dass der Dividend durch 2 × *n* Bits und der Divisor auf *n* Bits dargestellt wird. Die Operation liefert zwei Ergebnisse: den Quotient, der auf *n* Bits dargestellt ist, und den Rest, der ebenfalls durch *n* Bits dargestellt ist. Im Falle einer Vorzeichenkonvention werden der Dividend und der Divisor in einem komplementären Code dargestellt. Sowohl der Quotient als auch der Rest werden auch in einem komplementären Code dargestellt. Der Quotient von die Division der Regeln von die Zeichen folgt. Wichtig! Der Rest der Division ist in absoluten Zahlen niedriger als der absolute Wert der Divisor und hat das gleiche Vorzeichen wie der Dividend! Beispiel: -7: 3 gibt den Quotient -2 und den Rest -l an, dh -7 = (-2) × 3 + (-1).  Im Vergleich dazu besagt der Divisionssatz mit dem Rest der Arithmetik, dass der Rest eine positive Zahl sein muss. In der Arithmetik haben wir also (-7) = (-3) × 3 + 2, also wie der Quotient ist -3 und der Rest 2!  Die Division Operation signalisiert einen Divisor-Null Fehler (*Divide by zero*!). Wenn es nicht in die Größen passt, deutet es auf ein Überschwingen hin! Beispielsweise kann eine ungeteilte 1000 : 3 Division formal durchgeführt werden, da der Dividend durch 16 Bit und die Divisor durch 8 Bit dargestellt werden kann. Bei dieser Operation kommt es zu einem Überlauf, weil der Quotient 333 ist und nicht durch 8 Bits dargestellt werden kann! Der Rest der Division ist 1 und kann durch ein Byte dargestellt werden. Um eine solche Situation zu vermeiden, kann der Programmierer entscheiden, die 16-Bit-Operation durchzuführen, dh der Dividend durch 32 Bit und der Divisor durch 16 Bit dargestellt, sowie der Quotient und der Rest durch 16 Bits (333 passtest auf 16 Bits und wir werden keinen Überlauf haben).  *Vorzeichenlose Multiplikations- und Divisionsoperationen*  Eine vorzeichenlose Multiplikation von Ganzzahlen, die binär dargestellt sind, wird nach dem bekannten Algorithmus durchgeführt, wobei jedoch nur die Basis 2 verwendet wird. Der Einfachheit halber betrachten wir *n* = 4. Berücksichtigen Sie die Faktoren 11 und 13. Die Multiplikation in der Basis 2 ist: |
| |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  | 1 | 0 | 1 | 1 | × | |  |  |  |  | 1 | 1 | 0 | 1 |  | |  |  |  |  | 1 | 0 | 1 | 1 |  | |  |  |  | 0 | 0 | 0 | 0 |  |  | |  |  | 1 | 0 | 1 | 1 |  |  |  | |  | 1 | 0 | 1 | 1 |  |  |  |  | | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |  | | | |
| Observații   1. Înmulţirea generează o serie de produse parţiale care sunt adunate. Există câte un produs parţial pentru fiecare cifră a înmulţitorului. 2. Produsele parţiale sunt fie deînmulţitul, fie 0. 3. Produsele parţiale nenule pot fi adunate unul câte unul. Ele sunt obţinute deplasând deînmulţitul spre stânga cu câte o poziţie. 4. Înmulţirea a două numere de câte *n* biţi generează un produs care ocupă 2 × *n* biţi (motiv pentru care s-a impus restricţia de dimensiune amintită mai sus).   Împărțirea fără semn se efectuează, de asemenea, după regulile cunoscute ale aritmeticii. Să împărţim pe 147 la 11. Împărţirea în baza 2 este: |  | Bemerkungen   1. Die Multiplikation erzeugt eine Reihe von Teilprodukten, die gesammelt werden. Für jede Ziffer des Multiplikators gibt es ein Teilprodukt. 2. Teilprodukten werden der Multiplikator oder 0 sein. 3. Nicht-Null-Teilprodukte können einzeln hinzugefügt werden. Sie erhalten sie, indem Sie den Multiplikator um eine Position nach links bewegen. 4. Die Multiplikation von zwei *n*-Bit-Zahlen erzeugt ein Produkt, das 2 × *n* Bits belegt (weshalb die oben erwähnte Größenbeschränkung auferlegt wurde).   Eine Vorzeichenlose Division wird auch nach bekannten arithmetischen Regeln durchgeführt. Teilen wir 147 durch 11. Die Division in Basis 2 lautet: |
| |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | |  | 1 | 0 | 1 | 1 |  |  |  | 1 | 1 | 0 | 1 | |  | = | 1 | 1 | 1 | 0 |  |  |  |  |  |  | |  |  | 1 | 0 | 1 | 1 |  |  |  |  |  |  | |  |  | = | 0 | 1 | 1 | 1 |  |  |  |  |  | |  |  |  | 0 | 0 | 0 | 0 |  |  |  |  |  | |  |  |  | = | 1 | 1 | 1 | 1 |  |  |  |  | |  |  |  |  | 1 | 0 | 1 | 1 |  |  |  |  | |  |  |  |  | = | 1 | 0 | 0 |  |  |  |  | | | |
| Într-adevăr, 147 împărţit la 11 dă câtul 13 şi restul 4.  În ultimă instanţă, împărţirea în baza 2 se reduce la o succesiune de scăderi succesive combinate cu operaţii de deplasare.  Observații   1. Toţi operanzii implicaţi în operaţii sunt reprezentaţi în cod complementar, în conformitate cu cerinţele de dimensionare expuse mai sus; 2. Atât la înmulţirea cu semn cât şi la împărţirea cu semn, se respectă regula semnelor; 3. Pentru operaţia de împărţire, restul este în modul mai mic decât modulul împărţitorului, iar semnul restului este acelaşi cu semnul deîmpărţitului.   Algoritmii de înmulţire şi împărţire în cod complementar nu pot fi preluaţi direct de la cei fără semn, ca în cazul adunării şi scăderii. |  | Tatsächlich ergibt 147 geteilt durch 11 der Quotient 13 und der Rest 4.  Letztendlich wird die Division in Basis 2 auf eine Abfolge aufeinanderfolgender Verringerungen in Kombination mit Verschiebungsoperationen reduziert.  Bemerkungen   1. Alle Operanden, die an Operationen beteiligt sind, werden in einem komplementär Code gemäß den oben angegebenen Bemaßungsanforderungen dargestellt; 2. Sowohl dem Vorzeichenmultiplikation als auch der Vorzeichendivision folgt die Vorzeichenregel; 3. Für die Division Operation befindet sich der Rest im unteren Modus als der Divisor, und das Restzeichen ist dasselbe wie das Dividend.   Multiplikations- und Divisionsalgorithmen in komplementären Code können nicht direkt aus dem Vorzeichenlosen Code entnommen werden, wie im Fall von Addition und Subtraktion. |
| 3.8 Conversia la o locaţie de alte dimensiuni |  | 3.8 An einen anderen Speicherort konvertieren |
| Până acum am presupus că operanzii au lungimi fixe, aşa cum pretind regulile de derulare a operaţiilor. Dar ce-i de făcut atunci când, spre exemplu, trebuie să se convertească un cod complementar pe 8 biţi la unul pe 16 biţi? Sau dacă trebuie să reducem un număr reprezentat fără semn pe 16 biţi la unul similar pe 8 biţi?  De fapt este vorba despre patru operaţii:   * Extensia cu semn a unui cod complementar într-o locaţie mai mare; * Extensia cu zero a unui număr fără semn într-o locaţie mai mare; * Contracţia cu semn a unui cod complementar într-o locaţie mai mică; * Contracţia de zero a unui număr fără semn într-o locaţie mai mică.   Regulile de conversie sunt foarte simple, extensia cu semn înseamnă că în spaţiul suplimentar toți biții vor avea ca valoare valoarea bitului de semn al reprezentării care se converteşte. Extensia cu zero înseamnă că în spaţiul suplimentar toţi biţii vor avea valoarea zero. Iată câteva exemple: |  | Bisher haben wir angenommen, dass die Operanden feste Längen haben, wie es die Betriebsregeln von Operationen vorschreiben. Aber was tun, wenn beispielsweise ein komplementärer 8-Bit-Code in einen 16-Bit-Code konvertiert werden soll? Oder sollten wir eine vorzeichenlose 16-Bit-Zahl auf eine ähnliche 8-Bit-Zahl reduzieren?  In der Tat geht es um vier Operationen:   * Vorzeichenerweiterung eines komplementären Codes an einem größeren Speicherort; * Null-Erweiterung einer Vorzeichenlose Zahl an einer größeren Speicherort; * Vorzeichenkontraktion eines komplementären Codes an einem kleineren Speicherort; * Null-Kontraktion einer Vorzeichenlosen Zahl an einen kleineren Speicherort.   Die Konvertierungsregeln sind sehr einfach. Die Vorzeichenerweiterung bedeutet, dass in dem zusätzlichen Raum alle Bits den Wert des Vorzeichenbits der zu konvertierenden Darstellung bewerten. Die Null-Erweiterung bedeutet, dass in dem zusätzlichen Raum alle Bits Null sind. Hier einige Beispiele: |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | **8 biți** | **16 biți:**  **Extensie cu semn**  **(*Vorzeichenerweiterung*)** | **32 biți:**  **Extensie cu semn**  **(*Vorzeichenerweiterung*)** | **16 biți:**  **Extensie cu zero**  **(*Null-Erweiterung*)** | **32 biți:**  **Extensie cu zero**  **(*Null-Erweiterung*)** | | 80  1000 0000 | FF80  1111 1111 1000 0000 | FFFFFF80  1111 1111 1111 1111 1111 1111 1000 0000 | 0080  0000 0000 1000 0000 | 00000080  0000 0000 0000 0000 0000 0000 1000 0000 | | 28  0010 1000 | 0028  0000 0000 0010 1000 | 00000028  0000 0000 0000 0000 0000 0000 0010 1000 | 0028  0000 0000 0010 1000 | 00000028  0000 0000 0000 0000 0000 0000 0010 1000 | | 9A  1001 1010 | FF9A  1111 1111 1001 1010 | FFFFFF9A  1111 1111 1111 1111 1111 1111 1001 1010 | 009A  0000 0000 1001 1010 | 0000009A  0000 0000 0000 0000 0000 0000 1001 1010 | | 7F  0111 1111 | 007F  0000 0000 0111 1111 | 0000007F  0000 0000 0000 0000 0000 0000 0111 1111 | 007F  0000 0000 0111 1111 | 0000007F  0000 0000 0000 0000 0000 0000 0111 1111 | | - | 1020  0001 0000 0010 0000 | 00001020  0000 0000 0000 0000 0001 0000 0010 0000 | ---- | 00001020  0000 0000 0000 0000 0001 0000 0010 0000 | | - | 8088  1000 0000 1000 1000 | FFFF8088  1111 1111 1111 1111 1000 0000 1000 1000 | ---- | 00008088  0000 0000 0000 0000 1000 0000 1000 1000 | | | |
|  |  |  |
| Operaţiile de contracţie nu se pot executa întotdeauna. Spre exemplu, într-o locaţie pe 16 biţi există numărul -448 în baza 10, care în cod complementar se reprezintă FE40. Dorim să efectuăm o contracție la 8 biţi. Eliminând pur şi simplu primul octet, se obţine 40, adică numărul 64 în baza 10! Avem, evident, o situaţie de depăşire. Cu alte cuvinte, contracţiile (conversii prin îngustare) se pot executa numai dacă NU se provoacă pierderea de informaţie.  Contracţia cu semn se poate face numai dacă toţi biţii care se elimină coincid cu bitul de semn, adică cu primul bit care rămâne. Pentru contracţia fără semn, trebuie ca toți biţii care se elimină să fie zero. Tabelul următor prezintă câteva exemple. |  | Die Kontraktionsoperationen können nicht immer durchgeführt werden. Beispielsweise gibt es an einer 16-Bit-Stelle die Nummer -448 in der Basis 10, die im komplementären Code FE40 dargestellt ist. Wir möchten eine 8-Bit-Kontraktion durchführen. Durch einfaches Entfernen des ersten Bytes erhalten Sie 40, also die Zahl 64 in der Basis 10! Wir haben offensichtlich eine Situation der Überschwingen. Mit anderen Worten, Kontraktionen (einschränkende Konvertierungen) können nur ausgeführt werden, wenn KEINE Informationen verloren gehen.  Eine Vorzeichenkontraktion kann nur durchgeführt werden, wenn alle entfernten Bits mit dem Vorzeichenbit, dh dem ersten verbleibenden Bit, übereinstimmen. Für eine Vorzeichenloskontraktion müssen alle Bits, die gelöscht werden, Null sein. Die folgende Tabelle zeigt einige Beispiele. |
| |  |  |  | | --- | --- | --- | | **16 biți** | **8 biți: contracție cu semn**  **(*Vorzeichenkontraktion*)** | **8 biți: contracție cu zero**  **(*Null-Kontraktion*)** | | FF80  1111 1111 1000 0000 | 80  1000 0000 | Depășire! (*Überschwingen*!)  Se pierd 8 biți de 1  (*8 Bits von 1 gehen verloren*) | | 0028  0000 0000 0010 1000 | 28  0010 1000 | 28  0010 1000 | | FF9A  1111 1111 1001 1010 | 9A  1001 1010 | Depășire! (*Überschwingen*!)  Se pierd 8 biți de 1  (*8 Bits von 1 gehen verloren*) | | FE40  1111 1110 0100 0000 | Depășire! (*Überschwingen*!)  Se schimbă bitul de semn  (*Ändern des Vorzeichenbits*) | Depășire! (*Überschwingen*!)  Se pierd 8 biți de 1  (*8 Bits von 1 gehen verloren*) | | 0100  0000 0001 0000 0000 | Depășire! (*Überschwingen*!)  Se schimbă bitul de semn  (*Ändern des Vorzeichenbits*) | Depășire! (*Überschwingen*!)  Se pierde ultimul bit de 1  (*Das Letze* *Bit von 1 geht verloren*) | | 0088  0000 0000 1000 1000 | Depășire! (*Überschwingen*!)  Se schimbă bitul de semn  (*Ändern des Vorzeichenbits*) | 88  1000 1000 | | | |
|  |  |  |