Acesta este capitolul 7 — *Codificări de interes practic* — al ediţiei electronică a cărţii *Reţele de calculatoare*, publicată la Casa Cărţii de Ştiinţă, în 2008, ISBN: 978-973-133-377-9.

Cartea, integrală, poate fi descărcată gratuit de la adresa
`http://www.cs.ubbcluj.ro/~rlupsa/works/retele.pdf`

Capitolul 7

Codificări de interes practic

7.1. Probleme privind reprezentarea numerelor întregi

Până aici am privit un mesaj transmis printr-o reţea ca un şir de simboluri dintr-un alfabet finit. Între simbolurile ce alcătuiesc şirul se stabileşte o *ordine*, existând un prim element al şirului, un al doilea, etc. Transmisia elementelor se face în ordinea în care apar ele în şir, primul simbol transmis fiind cel care ocupă prima poziţie din şir. Până aici am considerat că transmisia unui şir de la un dispozitiv la altul conservă ordinea între elemente.

Aşa cum vom vedea însă în paragraful de faţă, din raţiuni legate de standardizarea reprezentării numerelor întregi, transmiterea unui şir nu conservă întotdeauna ordinea elementelor. În cele ce urmează, vom examina interferenţele între reprezentarea numerelor în calculator şi ordinea simbolurilor ce alcătuiesc un mesaj. Vom oferi cititorului o perspectivă, inspirată din [Cohen 1980] şi mai puţin întâlnită în alte lucrări, asupra relaţiilor dintre biţi, octeţi şi reprezentarea numerelor.

7.1.1. Reprezentări pe biţi

În paragraful de faţă vom face abstracţie de anumite complicaţii constructive ale sistemelor de calcul reale. Vom considera reprezentări pe biţi, ignorând deocamdată aspectele legate de gruparea biţilor în octeţi şi de faptul că, în memoria calculatorului, adresele identifică octeţi şi nu biţi.

Ca urmare, rugăm cititorul să uite, pentru moment, noţiunea de *octet* (*byte*).

7.1.1.1. Bitul

Pentru reprezentarea diverselor date, alegerea unui alfabet cu două simboluri este avantajoasă din două motive. Pe de o parte, este cel mai mic alfabet posibil, ca urmare alegerea unui alfabet cu două elemente aduce o anumită simplitate şi naturaleţe construcţiei matematice. Pe de altă parte, din punct de vedere practic, al construcţiei echipamentelor fizice, dispozitive cu două stări stabile sunt mult mai uşor de construit decât dispozitive cu mai multe stări.

În scris, cele două simboluri sunt notate în mod obişnuit cu 0 şi 1. Atragem atenţia că:

- Alegerea celor două simboluri utilizate, precum şi a corespondenţei dintre starea dispozitivului fizic şi simbolul asociat, poate fi făcută oricum. Odată însă făcută o alegere, aceasta trebuie respectată de toate entităţile implicate în comunicaţie.
- Numai în unele cazuri simbolurile au rol de cifră, adică au asociate valori numerice. Valoarea numerică a unui simbol este importantă doar dacă simbolul este interpretat ca număr sau intră în reprezentarea unui număr. În restul cazurilor, este important doar să existe simboluri distincte.

Un simbol dintr-un alfabet cu două elemente se numeşte *bit*, de la *binary digit* (rom. *cifră binară*).

7.1.1.2. Şiruri de biţi

În cadrul unui şir de biţi, avem nevoie să identificăm fiecare bit al şirului. Pentru aceasta, se stabileşte o *ordine* a biţilor: avem un prim bit, un al doilea bit, etc. Trebuie remarcat însă că ordinea este o convenţie: nu există o legătură directă între ordinea convenţională a unui şir de biţi şi amplasamentul dispozitivelor fizice care memorează acei biţi. Numărul de ordine al unui bit, în cadrul acestei ordini convenţionale, se numeşte în mod obişnuit *poziţia* (sau, eventual, *adresa* sau *deplasamentul*) bitului. Numerotarea poziţiilor se face de obicei începând de la 0 sau de la 1; în cele ce urmează vom utiliza numerotarea de la 0.

La transmiterea unui şir de biţi, este natural ca primul bit transmis, considerând ordinea cronologică, să fie primul bit al şirului (poziţia 0). La memorarea unui şir într-o memorie cu acces direct (memorie RAM sau fişier pe disc), este natural ca în celula cu adresa cea mai mică, dintre celulele alocate şirului, să fie plasat primul bit al şirului (bitul de pe poziţia 0). În acest fel, *primul bit* al unui şir înseamnă, simultan, bitul de pe poziţia (convenţională) 0, bitul transmis primul (cronologic) şi bitul memorat la adresa cea mai mică.

7.1.1.3. Reprezentarea pe biţi a numerelor întregi

Reprezentarea numerelor naturale prin şiruri de biţi se bazează pe ceea ce matematicienii numesc *reprezentare poziţională în baza 2*, pe care o presupunem cunoscută. În reprezentarea într-o bază de numeraţie, distingem cifra unităţilor, având ponderea $2^0 = 1$, cifra de pondere $2^1 = 2$ (în baza zece s-ar numi cifra zecilor; pentru baza 2 nu avem un nume), cifra de pondere $2^2 = 4$, etc.

Există două alegeri posibile cu privire la legătura dintre ponderile cifrelor în număr şi poziţiile lor în şir:

1. *Primul bit al şirului este cifra de pondere maximă.* Aceasta alegere este identică celei utilizate în scrierea numerelor în limbile „obişnuite“ (indo-europene), cu scriere de la stânga spre dreapta. Se mai numeşte *big endian*.

   În această schemă de reprezentare, un şir de biţi $b_0 b_1 \ldots b_{n-1}$ reprezintă numărul

   $$b_0 \cdot 2^{n-1} + b_1 \cdot 2^{n-2} + \ldots + b_{n-1} \cdot 2^0.$$

2. *Primul bit al şirului este cifra de pondere 1.* Această reprezentare este asemănătoare scrierii numerelor în limbile semite (araba şi ebraica, cu scriere de la dreapta spre stânga şi unde numerele sunt scrise tot cu cifra unităţilor în dreapta). Se mai numeşte *little endian*.

   Această alegere are avantajul unei scrieri mai simple a relaţiei dintre valoarea numărului şi şirul de biţi: valoarea unui număr reprezentat pe n biţi este

   $$b_0 \cdot 2^0 + b_1 \cdot 2^1 + \ldots + b_{n-1} \cdot 2^{n-1}.$$

Fiind două scheme de reprezentare distincte, dacă un sistem transmite un număr în reprezentare *little endian*, iar celălalt sistem interpretează şirul de biţi primit ca fiind într-o reprezentare *big endian*, receptorul „înţelege“ alt număr decât cel transmis de emiţător. Ca urmare, orice protocol care specifică transmitere binară a numerelor trebuie să precizeze dacă se utilizează o reprezentare *little endian* sau una *big endian*.

Exemplul 7.1: Fie şirul de biţi 11001, în care am scris primul bit (în sensul din § 7.1.1.2) pe poziţia cea mai din stânga.

Dacă acest şir este reprezentarea *big endian* a unui număr, numărul respectiv este 25. Dacă reprezentarea a fost făcută în format *little endian*, numărul este 19.

Este important de remarcat că distincţia dintre schema de reprezentare *big endian* si schema *little endian* există numai acolo unde pe de o parte avem o ordine a a biţilor dată de adresele lor în memorie sau de ordinea cronologică la transmiterea lor prin mediul fizic, iar pe de altă parte fiecare bit are o anumită pondere în reprezentarea unui număr întreg.

7.1.2. Reprezentări pe octeţi

În paragraful precedent, am ignorat în mod deliberat noţiunea de *octet* (*byte*) şi am presupus că, în memorie, fiecare bit ar avea o adresă individuală. Vom studia, în continuare, problemele legate de gruparea, în cadrul sistemelor de calcul reale, a biţilor în octeţi şi de faptul că, în general, ordinea biţilor în octeţi nu este vizibilă programatorului.

7.1.2.1. Octeţi

În memoria calculatoarelor, biţii sunt grupaţi în grupuri de dimensiune fixă, în mod obişnuit câte 8 biţi. Un astfel de grup se numeşte *octet* (denumire intrată pe filieră franceză) sau *bait* (adaptare a englezescului *byte*).

Un octet poate fi privit în două moduri distincte:

- ca un şir de 8 biţi,
- ca un număr întreg cuprins între 0 şi 255.

Echivalenţa între aceste două moduri de-a privi un octet este o problemă ce necesită multă atenţie. Dacă identificăm biţii după ponderile lor, există o corespondenţă biunivocă între un astfel de grup de 8 biţi şi un număr întreg între 0 şi 255. Dacă însă identificăm biţii după poziţia lor în şirul de 8 biţi (bitul 0, bitul 1,..., bitul 7), atunci corespondenţa biunivocă între număr şi şir de biţi există doar după ce am stabilit o corespondenţă între poziţia unui bit în şir şi ponderea sa (*big endian*, *little endian* sau eventual o corespondenţă mai complicată).

După modul în care se identifică biţii unui octet în definiţia operaţiilor efectuate de diverse componente ale unui sistem de calcul, operaţiile se pot împărţi în trei categorii:

1. *Operaţii pentru care biţii se identifică după pondere* sau, echivalent, octetul este privit ca număr. Aici se încadrează operaţiile aritmetice şi operaţiile de deplasare pe biţi. De remarcat că operaţiile *deplasare la stânga* (engl. *shift left*), respectiv *deplasare la dreapta* (engl. *shift right*) pot fi descrise în termeni de operaţii aritmetice: deplasarea la stânga este o înmulţire cu 2, iar deplasarea la dreapta este o împărţire la 2. În acest context, „spre stânga“ şi „spre dreapta“ înseamnă spre poziţiile cu pondere mai mare, respectiv mai mică, neavând nici o legătură cu

primele sau cu ultimele poziţii. Aceste operaţii sunt efectuate de unitatea aritmetică din microprocesorul calculatorului.

2. *Operaţii pentru care biţii sunt identificaţi după numărul lor de ordine* sau, echivalent, octetul este privit ca un şir arbitrar de biţi, fără a avea asociată o valoare numerică. Aici intră transmiterea bit cu bit (transmitere serială) a unui octet. Această operaţie este efectuată de placa de reţea şi de alte adaptoare seriale (de exemplu, adaptoarele USB). Tot aici s-ar încadra, dacă ar exista, o operaţie de obţinere sau de modificare a unui bit (al octetului) identificat prin numărul său de ordine. O asemenea operaţie nu este oferită, în mod normal, într-un sistem de calcul — nu există o instrucţiune care să extragă, de exemplu, bitul numărul 5 dintr-un octet.
3. *Operaţii care pot fi definite fie identificând biţii după ponderea lor, fie identificând biţii după numărul lor de ordine.* În această categorie se încadrează transmiterea unui octet ca un tot unitar, verificarea egalităţii a doi octeţi şi operaţiile logice pe bit (*şi*, *sau*, *sau exclusiv* şi negaţia).

   Pentru oricare dintre aceste operaţii, dacă definim operaţia identificând biţii după numărul lor de ordine, efectul ei asupra valorii numerice a octetului nu depinde de corespondenţa aleasă între poziţiile biţilor şi ponderile lor.

În aceste condiţii, în interiorul unui calculator, biţii din cadrul unui octet sunt identificaţi după ponderea lor. În construcţia calculatorului, proiectantul are grijă ca atunci când un bit având, într-un modul al calculatorului, o anumită pondere este transferat către alt modul al calculatorului, să ajungă acolo pe o poziţie cu aceeaşi pondere.

La transmisia unui octet între două sisteme de calcul, mecanismele de transmisie sunt astfel construite încât să transmită valoarea octetului. Întrucât, prin mediul fizic al reţelei, biţii sunt transmişi secvenţial, biţii unui octet sunt aici identificaţi prin numărul lor de ordine in cadrul transmisiei. Pentru a păstra valoarea octetului în timpul transmisiei prin mediul reţelei, corespondenţa dintre numărul de ordine al unui bit şi ponderea sa (*little endian* sau *big endian*) trebuie să facă parte din specificaţiile protocolului de nivel fizic al reţelei.

Numerotarea biţilor unui octet intervine, de asemenea, în descrierea unor scheme de reprezentare a datelor unde un număr este reprezentat pe un grup de biţi ce nu formează un număr întreg de octeţi. Este cazul schemelor de reprezentare pentru structuri de date ce conţin câmpuri de 1 bit, 2 biţi, 12 biţi, etc. Şi aici este necesar să se specifice dacă numerotarea biţilor este *little*

endian sau *big endian*. Mai multe detalii despre astfel de reprezentări vor fi studiate în § 7.1.2.4.

7.1.2.2. Şiruri de octeţi

Ca şi în cazul biţilor (vezi § 7.1.1.2), şi cu octeţii putem construi şiruri. În cadrul unui şir de octeţi, octeţii sunt aşezaţi într-o ordine, existând un prim octet (numerotat ca octetul 0), al doilea octet (poziţia 1), etc. La transmisia printr-o legătură în reţea, primul octet al şirului este, cronologic, primul octet transmis. La memorare, primul octet este cel memorat la adresa cea mai mică.

În virtutea celor două moduri de-a privi un octet, un şir de n octeţi poate fi, la rândul lui, privit ca:

- un şir de $8n$ biţi,
- un şir de n numere, fiecare cuprins între 0 şi 255.

Pentru a putea privi un şir de n octeţi ca un şir de $8n$ biţi, este necesar să avem o numerotare, bine definită, a biţilor în cadrul unui octet. Rezultă un şir de biţi în care între poziţia p_B a unui bit în şirul de $8n$ biţi, poziţia p_{BO} a bitului în cadrul octetului în care se găseşte şi poziţia p_O a acelui octet în şirul de octeţi are loc relaţia:

$$p_B = 8 \cdot p_O + p_{BO}. \tag{7.1}$$

Relaţia de mai sus are această formă simplă dacă se utilizează numerotare de la 0; pentru numerotarea de la 1, forma relaţiei e mai complicată.

Transmiterea unui şir de octeţi printr-o conexiune, precum şi memorarea şirului într-un fişier pe disc urmată de citirea lui înapoi în memorie, păstrează ordinea şi valorile octeţilor din şir. Valorile octeţilor sunt păstrate dacă privim octeţii ca numere între 0 şi 255; dacă privim octeţii ca şiruri de 8 biţi, valorile octeţilor se păstrează numai dacă pe ambele sisteme utilizăm aceeaşi corespondenţă între numerele de ordine şi ponderile biţilor.

7.1.2.3. Reprezentarea numerelor pe un număr întreg de octeţi

Cel mai mare număr ce poate fi reprezentat pe un octet este 255, ceea ce este mult prea puţin pentru majoritatea aplicaţiilor. Pentru a putea reprezenta numere din intervale mai largi, sunt necesare scheme de reprezentare pe mai mult de 8 biţi. Schemele cele mai simple sunt cele care utilizează un număr întreg de octeţi; acestea vor fi prezentate în continuare. Schemele de reprezentare ce utilizează şiruri de biţi ce nu formează neapărat un număr întreg de octeţi vor fi studiate în § 7.1.2.4.

Deoarece un octet are valoarea între 0 şi 255, putem considera fiecare octet ca fiind o cifră în baza 256. Reprezentarea unui număr printr-un şir de octeţi se face ca reprezentare poziţională în baza 256. Există două reprezentări posibile:

- *little endian*: primul octet are ponderea 1, al doilea octet are ponderea 256, al treilea octet are ponderea $256^2 = 65536$, etc.
- *big endian*: primul octet are ponderea 256^{n-1} (unde n este numărul de octeţi ai reprezentării), al doilea octet are ponderea 256^{n-2} ş. a. m. d., penultimul octet are ponderea 256, iar ultimul octet are ponderea 1.

Reamintim că prin *primul octet* înţelegem octetul care este transmis primul, în ordine cronologică, de la un dispozitiv la altul şi, totodată, octetul memorat la adresa cea mai mică.

Exemplul 7.2: Descriem mai jos reprezentarea numărului 300 în schemele de reprezentare *little endian* şi *big endian*, pe 2 şi pe 4 octeţi. Pentru fiecare dintre aceste patru scheme de codificare, este dat şirul de octeţi ce reprezintă numărul 300.

poziţie octet (nr. ordine)	Valorile octeţilor pentru diverse reprezentări			
	2 octeţi big endian	2 octeţi little endian	4 octeţi big endian	4 octeţi little endian
0	1	44	0	44
1	44	1	0	1
2	—	—	1	0
3	—	—	44	0

De exemplu, în cadrul reprezentării pe 2 octeţi în format *big endian*, valoarea numărului reprezentat se regăseşte ca valoarea octetului 0 înmulţită cu 256 plus valoarea octetului 1, anume: $1 \cdot 256 + 44 = 300$. În cadrul reprezentării pe 4 octeţi în format *little endian*, octetul 0 are ponderea $256^0 = 1$, octetul 1 are ponderea $256^1 = 256$, octetul 2 are ponderea $256^2 = 65536$, iar octetul 3 are ponderea $256^3 = 16218368$. Valoarea numărului reprezentat este

$$44 \cdot 1 + 1 \cdot 256 + 0 \cdot 256^2 + 0 \cdot 256^3 = 300$$

Unitatea aritmetică a calculatorului poate efectua operaţii aritmetice asupra numerelor reprezentate pe 2 octeţi sau, pentru unele calculatoare, pe 4 sau 8 octeţi. Pe unele calculatoare, unitatea aritmetică lucrează cu numere reprezentate după sistemul *big endian*; pe alte calculatoate, unitatea

aritmetică cere reprezentare *little endian*. După acest criteriu, calculatoarele ale căror unităţi aritmetice lucrează cu numere reprezentate pe mai mult de un octet se împart în calculatoare *little endian* şi calculatoare *big endian*.

Variabilele de tip întreg, în majoritatea limbajelor de programare, sunt reprezentate pe 2, 4 sau 8 octeţi, în ordinea fixată de unitatea aritmetică.

Este posibilă utilizarea, pentru anumite variabile întregi, a unei reprezentări diferite de cea a unităţii aritmetice. De asemenea, se pot utiliza reprezentări pe mai mulţi octeţi decât permite unitatea aritmetică. La manipularea acestor variabile, programatorul trebuie să aibă în vedere că operaţiile aritmetice „normale“ fie nu pot fi executate deloc, fie nu se efectuează corect asupra lor. Astfel de numere se manipulează, de obicei, prelucrând separat fiecare octet.

La memorarea pe disc sau la transmiterea printr-o conexiune, trebuie stabilit printr-un standard dacă se utilizează un format *little endian* sau *big endian*, precum şi numărul de octeţi pe care se reprezintă fiecare număr memorat sau, respectiv, transmis. Majoritatea protocoalelor pentru Internet prevăd formate *big endian* pentru numerele întregi transmise. Multe dintre formatele de fişiere prevăd formate *little endian*. Există şi formate (de exemplu, formatul TIFF pentru imagini, formatele UTF-16 şi UTF-32 pentru texte) care permit emiţătorului să aleagă formatul dorit şi prevăd un mecanism prin care emiţătorul informează receptorului despre alegerea făcută.

Dacă formatul de pe disc sau de pe conexiune coincide cu formatul unităţii aritmetice locale, un program poate transfera direct şiruri de octeţi între o variabilă întreagă locală şi fişierul de pe disc sau, respectiv, conexiunea spre celălalt calculator. Dacă formatul de pe disc sau de pe conexiune este invers faţă de cel local, un program care transferă date trebuie să inverseze ordinea octeţilor imediat înainte de scrierea pe disc sau de trimiterea pe conexiune, precum şi imediat după citirea de pe disc sau recepţionarea de pe conexiune.

7.1.2.4. Reprezentarea numerelor pe un şir arbitar de biţi

Ne vom ocupa în continuare de metode de reprezentare, pentru numere întregi, în care biţii ce intră în reprezentarea unui număr nu formează neapărat un număr întreg de octeţi. O astfel de schemă este o generalizare a schemei prezentate în paragraful precedent.

O astfel de metodă de reprezentare se bazează pe reprezentarea numerelor în baza 2 (vezi şi § 7.1.1.3). Pentru ca o astfel de schemă să fie complet definită, este necesar să fie stabilită (standardizată) corespondenţa dintre poziţia fiecărui bit din reprezentare şi ponderea asociată. În descrierea

unei astfel de scheme de reprezentare, trebuie precizate trei lucruri:

- dacă reprezentarea numărului prin şirul de biţi se face după metoda *big endian* sau *little endian*;
- dacă numerotarea biţilor în cadrul fiecărui octet se face începând de la bitul de pondere 1 (adică valoarea octetului este reprezentată după schema *little endian*) sau începând de la bitul de pondere 128 (adică valoarea octetului este reprezentată după schema *big endian*).
- dacă numerotarea biţilor în cadrul şirului de biţi se face după relaţia (7.1) sau după o altă metodă.

Într-o schemă de reprezentare „raţională“, la primele două puncte se utilizează fie formatul *big endian* pentru amândouă, fie formatul *little endian* pentru amândouă, iar la punctul al treilea se utilizează relaţia (7.1).

Rezultă astfel două metode coerente:

- *big endian.* În această metodă, numerotarea biţilor începe cu cel mai semnificativ bit al primului octet, iar după cel mai puţin semnificativ bit al primului octet urmează cel mai semnificativ bit al celui de-al doilea octet. Orice număr, indiferent pe câţi biţi s-ar reprezenta, se reprezintă în format *big endian.*
- *little endian.* În această metodă, numerotarea biţilor începe cu cel mai puţin semnificativ bit al primului octet, iar după cel mai semnificativ bit al primului octet urmează cel mai puţin semnificativ bit al celui de-al doilea octet. Orice număr, indiferent pe câţi biţi s-ar reprezenta, se reprezintă în format *little endian.*

În cadrul acestor două metode, dacă reprezentăm un număr folosind un şir de biţi ce formează un număr întreg de octeţi, formatele de reprezentare rezultate coincid cu formatele de reprezentare pe octeţi studiate în paragraful precedent.

Exemplul 7.3: Considerăm o schemă de reprezentare pentru două numere întregi, a şi b, în care a se reprezintă pe 4 biţi şi b se reprezintă pe 12 biţi. În total avem $4 + 12 = 16$ biţi, adică 2 octeţi.

Dacă alegem metoda *big endian*, schema de reprezentare va utiliza cei mai semnificativi 4 biţi ai primului octet pentru a-l reprezenta pe a, ceilalţi 4 biţi ai primului octet vor fi cei mai semnificativi 4 biţi ai lui b, iar cel de-al doilea octet va conţine cei mai puţin semnificativi 8 biţi din b. Această schemă de reprezentare este ilustrată în figura 7.1, cu valori concrete $a = 11$ şi $b = 300$.

Dacă alegem metoda *little endian*, schema de reprezentare va utiliza cei mai puţin semnificativi 4 biţi ai primului octet pentru a-l reprezenta pe a, ceilalţi 4 biţi ai primului octet vor fi cei mai puţin semnificativi 4 biţi ai lui b,

iar cel de-al doilea octet va conţine cei mai semnificativi 8 biţi din b. Această schemă de reprezentare este ilustrată în figura 7.2, cu valori concrete $a = 11$ şi $b = 300$.

b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
1	0	1	1	0	0	0	1	0	0	1	0	1	1	0	0

(a) Reprezentarea privită ca şir de biţi

Nr. octet	Valoare (binar)								Valoare (zecimal)
	c_0	c_1	c_2	c_3	c_4	c_5	c_6	c_7	
0	1	0	1	1	0	0	0	1	177
1	0	0	1	0	1	1	0	0	44

(b) Valorile octeţilor. La scrierea valorilor biţilor în octet (coloana din mijloc) s-a utilizat convenţia obişnuită, de-a scrie cifrele mai semnificative în stânga.

Figura 7.1: Reprezentare *big endian* pentru numărul 11 pe 4 biţi urmat de numărul 300 reprezentat pe 12 biţi (exemplul 7.3).

b_0	b_1	b_2	b_3	b_4	b_5	b_6	b_7	b_8	b_9	b_{10}	b_{11}	b_{12}	b_{13}	b_{14}	b_{15}
1	1	0	1	0	0	1	1	0	1	0	0	1	0	0	0

(a) Reprezentarea privită ca şir de biţi

Nr. octet	Valoare (binar)								Valoare (zecimal)
	c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0	
0	1	1	0	0	1	0	1	1	203
1	0	0	0	1	0	0	1	0	18

(b) Valorile octeţilor. La scrierea valorilor biţilor în octet (coloana din mijloc) s-a utilizat convenţia obişnuită, de-a scrie cifrele mai semnificative în stânga.

Figura 7.2: Reprezentare *little endian* pentru numărul 11 pe 4 biţi urmat de numărul 300 reprezentat pe 12 biţi (exemplul 7.3).

Un alt exemplu în care avem de-a face cu numere reprezentate pe şiruri arbitrare de biţi este legat de aşa-zisa *codificare în baza 64*, descrisă în § 7.4.2.

7.1.3. Probleme privind reprezentarea lungimii şirurilor

Oridecâteori se transmite un şir de obiecte, este necesar ca receptorul să poată determina numărul de obiecte transmise. Acest lucru este valabil

indiferent de natura obiectelor: biţi, octeţi, cifre zecimale, caractere ale unui text, numere în cadrul unui şir de numere.

Există trei metode de a face ca receptorul să poată determina numărul de obiecte ce-i sunt transmise:

- *Numărul de obiecte este fixat.* În acest caz, indiferent de valorile datelor ce se transmit, numărul de obiecte transmise este acelaşi. Metoda este utilizată frecvent la memorarea unui şir în memoria RAM sau pe disc, deoarece permite alocarea de la început a memoriei pentru reprezentarea lui şi permite accesul direct la datele memorate după şirul în discuţie. Dezavantajul principal al metodei este acela că dimensiunea fixată trebuie astfel aleasă încât să fie suficientă în orice caz ce poate să apară la execuţie. De asemenea, trebuie să existe o valoare potrivită pentru poziţiile „neutilizate“ din şir; de exemplu, în reprezentarea numerelor, poziţiile cele mai semnificative se completează cu zerouri.

  Deoarece la transmisia printr-o conexiune nu se poate pune problema accesului direct (adică altfel decât secvenţial) la date, metoda este puţin utilizată în transmisia datelor prin reţea. Şiruri de lungime fixă se utilizează la reprezentarea binară a numerelor.
- *Numărul de obiecte este transmis separat, în faţa şirului.* Această metodă uşurează munca receptorului, care ştie exact ce să astepte şi poate aloca memorie pentru recepţionarea datelor. În schimb, munca emiţătorului este complicată prin faptul că acesta trebuie să cunoască de la început numărul de obiecte din şir. Acest fapt face metoda inaplicabilă în anumite situaţii.

  Transmiterea de la început a numărului de obiecte este utilizată, de exemplu, de protocolul *HTTP* (§ 11.3.2) la transmiterea, de către server, a conţinutului paginii cerute de client. Serverul transmite întâi numărul de octeţi ai paginii şi apoi şirul de octeţi ce formează pagina.
- *După ultimul obiect din şirul propriu-zis, se transmite o valoare specială, cu rol de* terminator. Această metodă uşurează munca emiţătorului, care poate să înceapă transmiterea şirului înainte de-a şti câte elemente are, în schimb îngreunează munca receptorului, care trebuie să citească elementele şirului unu câte unu şi să verifice dacă nu a întâlnit terminatorul. De asemenea, trebuie fixată valoarea terminatorului, care trebuie să fie o valoare reprezentabilă în formatul pentru element, dar care nu apare niciodată ca valoare a unui element valid.

  Metoda este utilizată frecvent în transmiterea unui şir de caractere. Rolul de terminator poate fi acordat caracterului *null* (caracterul cu codul ASCII zero), caracterului *newline* (sfârşit de rând), caracterului

spaţiu, etc. Orice alegere s-ar face, caracterul sau caracterele astfel alese pentru a marca sfârşitul unui şir nu pot să apară în şirul propriu-zis. Ca urmare, transmiterea unui fişier cu conţinut arbitrar (şir de octeţi cu valori arbitrare) nu se poate face prin metoda cu terminator (decât dacă un octet al fişierului se codifică pe mai mult de 8 biţi).

7.1.4. Alte metode de reprezentare a numerelor întregi

Schemele de reprezentare (formatele) pentru numere întregi, studiate în § 7.1.2.3 şi § 7.1.2.4, se numesc formate binare. Pe lângă formatele binare, pentru reprezentarea numerelor întregi se mai utilizează următoarele tipuri de formate:

- Formatul *text*. În cadrul acestui format, reprezentarea numărului este un text format din caracterele corespunzătoare cifrelor reprezentării zecimale (obişnuite) a numărului.
- Formatul *binar-zecimal*, numit şi *BCD* din engl. *binary coded decimal*. În cadrul acestui format, numărul este reprezentat mai întâi în baza 10, iar apoi fiecare cifră zecimală este reprezentată pe 4 biţi conform metodelor din § 7.1.2.4.

Descriem puţin mai pe larg reprezentarea numerelor în format text, deoarece o astfel de reprezentare se utilizează frecvent în protocoalele în reţea. Motivul principal al utilizării formatului text este uşurinţa depanării aplicaţiilor ce utilizeaza astfel de protocoale: comunicaţia poate fi înregistrată într-un fişier şi examinată cu un program obişnuit pentru vizualizarea fişierelor text.

În format text, se utilizează convenţiile de reprezentare a numerelor în textele scrise: se începe cu cifra cea mai semnificativă, numărul de cifre este variabil (depinde de valoarea numărului) şi prima cifră (cea mai semnificativă) scrisă nu este zero, cu excepţia cazului numărului 0 care se reprezintă ca o singură cifră zero.

Fiecare cifră se reprezintă ca un caracter, fiind necesară mai departe o schemă de reprezentare a textelor (vezi § 7.2). În cazul codificării ASCII, reprezentarea fiecărei cifre ocupă exact un octet. De remarcat însă că, în acest caz, cifra 0 nu se reprezintă ca un octet cu valoarea 0, ci ca un octet având ca valoare codul ASCII pentru caracterul „`0`“; acesta este 48 (sau, echivalent, 30_{16}).

Deoarece lungimea reprezentării este variabilă, este necesar să fie transmisă sub o formă sau alta informaţia privind lungimea reprezentării numărului (numărul de cifre). În acest scop, în reprezentările text, numerele sunt separate de obicei prin spaţii, caractere *tab*, caractere *newline* etc.

Exemplul 7.4: Redăm mai jos reprezentările text ASCII terminat cu spaţiu, BCD *big endian* pe 4 octeţi şi BCD *little endian* pe 4 octeţi, pentru numărul 300. Valorile octeţilor sunt scrise în baza 2, bitul cel mai semnificativ fiind scris în stânga.

poziţie octet (nr. ordine)	Valorile octeţilor		
	text ASCII	BCD *big endian*	BCD *little endian*
0	00110011	00000000	00000000
1	00110000	00000000	00000011
2	00110000	00000011	00000000
3	00100000	00000000	00000000

7.2. Codificarea textelor

Prin text înţelegem aici un text scris în limbaj natural sau într-un limbaj de programare, fără formatare avansată.

Un text este văzut în general ca o succesiune de *caractere*. Caracterele sunt în principal literele din alfabetul limbii în care este scris textul, semne de punctuaţie, cifre şi diferite alte semne grafice. Nu se face distincţie între diferitele variante de-a scrie o aceeaşi literă (litere „normale“, cursive („italice“), adline („bold“), etc).

Pe lângă caracterele grafice, descrise mai sus, sunt definite *caractere de control*, având rolul de a marca puncte (locuri) în cadrul textului sau fragmente din text. Utilizări ale caracterelor de control sunt, de exemplu, trecerea la rând nou sau interzicerea trecerii la rând nou (ruperea rândului) într-un anumit punct.

Un aspect discutabil legat de alegerea setului de caractere este dacă o literă cu un semn diacritic este cazul să fie caracter distinct faţă de litera simplă din care provine sau să fie format din caracterul corespunzător literei respective fără semne diacritice şi un caracter de control care să marcheze semnul diacritic adăugat. În aceeaşi idee, s-ar putea face şi distincţia dintre literele mari (majuscule) şi literele mici (minuscule) corespunzătoare tot pe baza unor caractere de control cu rol de modificator.

Operaţiile efectuate asupra textelor, care trebuie să fie permise de codificarea aleasă, sunt în principal următoarele:

- afişarea textului;
- concatenarea unor texte sau alte operaţii de sinteză;

- căutarea unui cuvânt, extragerea unor cuvinte sau unor fragmente de text şi diverse alte operaţii de analiză a textului;
- sortarea alfabetică.

De notat că regulile de sortare alfabetică sunt complexe şi depind de limbă. De exemplu, în limba română, literele cu diacritice sunt considerate imediat după literele fără diacritice. Următoarele cuvinte sunt sortate alfabetic: *sac*, *suc*, *şiret*; de notat că orice cuvânt ce începe cu *ş* este sortat după toate cuvintele ce încep cu *s*. În franceză, însă, literele cu diacritice sunt considerate, în prima fază, echivalente cu cele fără diacritice, intervenind în ordinea alfabetică doar pentru cuvinte care diferă doar prin diacritice. Exemplu: *été*, *être*, *étude*; de notat că apar amestecate cuvinte ce încep cu *é* şi *ê*.

Majoritatea codificărilor sunt bazate pe reprezentarea una după alta a literelor (caracterelor) ce formează cuvintele textului.

Codificările caracterelor sunt de obicei descrise în două etape. În prima etapă, fiecărui caracter îi este asociat un număr întreg pozitiv, numit *codul caracterului*. În a doua etapă, fiecărui cod de caracter îi este asociată o codificare ca şir de biţi sau ca şir de octeţi.

Pentru o schemă de codificare trebuie aşadar specificate trei elemente:

- setul de caractere;
- numerotarea (codificarea) caracterelor;
- reprezentarea pe biţi sau pe octeţi a codurilor caracterelor.

7.2.1. Codificarea ASCII

Codificarea (codul) ASCII este codificarea cea mai des întâlnită pentru texte.

Setul de caractere cuprinde 128 caractere dintre care:

- 33 de caractere de control;
- caracterul *spaţiu* (considerat de unii ca fiind caracter imprimabil şi de alţii ca fiind caracter de control);
- 94 de caractere imprimabile, cuprinzând: 52 de litere (cele 26 litere ale alfabetului latin, cu cele două forme, majuscule şi minuscule) cele 10 cifre zecimale şi un număr de 32 de semne de punctuaţie şi alte simboluri.

Codurile asociate caracterelor ASCII sunt cuprinse între 0 şi 127, caracterele de control primind codurile 0–31 şi 127, spaţiul are codul 32, iar

(celelalte) caractere imprimabile au codurile cuprinse între 33 şi 126. Pentru a uşura sortarea alfabetică, codurile sunt grupate astfel:

- literele mari de la 65 (41 hexa) pentru `A` la 90 (5A hexa) pentru `Z`;
- literele mici de la 97 (61 hexa) pentru `a` la 122 (7A hexa) pentru `z`;
- cifrele de la 48 (30 hexa) pentru `0` la 57 (39 hexa) pentru `9`.

De remarcat şi că diferenţa dintre codul oricărei litere mici şi codul literei mari corespunzătoare este 32 (20 hexa).

Pentru reprezentarea unui caracter ASCII sunt necesari doar 7 biţi, însă cel mai adesea un caracter ASCII se reprezintă pe un octet, al cărui cel mai semnificativ bit este întotdeauna 0.

Datorită faptului că pe de o parte caracterele ASCII se reprezintă pe un octet, iar pe de altă parte că dintre caracterele de control multe nu sunt utilizate deloc în majoritatea aplicaţiilor, rămân multe coduri reprezentabile (cca. 140) care nu sunt utilizate. Se poate extinde setul de caractere, asociind noilor caractere coduri între 128 şi 255 sau coduri între 0 şi 31 a căror caractere corespunzătoare nu sunt folosite efectiv. Toate aceste codificări rezultate se numesc generic *seturi ASCII extinse*.

7.2.2. Codificările ISO-8859

ISO-8859 este o familie de coduri, construite toate ca extensii (alternative) ale codificării ASCII.

Fiecare cod din familie cuprinde câte 256 caractere: cele 128 caractere ASCII, plus 128 de caractere alese pentru a acoperi alfabetul câte unui grup de limbi. Limbile acoperite de câteva dintre codificările ISO-8859 sunt:

- ISO-8859-1, alfabetul latin pentru limbile din vestul Europei;
- ISO-8859-2, alfabetul latin pentru limbile din estul Europei;
- ISO-8859-5, alfabetul chirilic;
- ISO-8859-6, alfabetul arab;
- ISO-8859-7, alfabetul grecesc;
- ISO-8859-8, alfabetul ebraic.

Codurile asociate caracterelor sunt codurile din codificarea ASCII pentru cele 128 de caractere din setul ASCII şi numere de la 128 la 255 pentru caracterele suplimentare.

Reprezentarea pe octeţi pentru un text ISO-8859-n se face cu câte un octet pentru fiecare caracter, octetul conţinând codul caracterului.

Fiecare cod din familie este extensie a codului ASCII în sensul că mulţimea caracterelor din fiecare astfel de cod include mulţimea caracterelor

Caracter	Cod (hexa)	Caracter	Cod (hexa)
Ă	C3	ă	E3
Â	C2	â	E2
Î	CE	î	EE
Ş	AA	ş	BA
Ţ	DE	ţ	FE

Tabelul 7.1: Caracterele cu diacritice din alfabetul limbii române şi codificările ISO-8859-2 corespunzătoare

ASCII şi codurile asociate caracterelor comune cu setul ASCII coincid cu codurile ASCII. Ca urmare, un text ASCII este întotdeauna interpretat corect ca text ISO-8859-n. Pe de altă parte, un text scris în ISO-8859-n şi interpretat ca ISO-8859-m va fi evident interpretat greşit.

Ordinea numerică a codurilor din oricare dintre codificările ISO-8859 este diferită de ordinea alfabetică. În general, în ordinea alfabetică, literele cu diacritice îşi au locul imediat lângă literele similare fără diacritice; în codificările ISO-8859-1 sau ISO-8859-2, de exemplu, literele cu diacritice au codurile mai mari de 128 în vreme ce literele fără diacritice au coduri între 65 şi 123.

7.2.3. Codificările Unicode

Unicode este un set de caractere ce se doreşte să cuprindă litere din toate scrierile de pe Pământ. Numărul de caractere din unicode este limitat, datorită modurilor de codificare definite, la aproximativ un milion (mai exact, la $110000_{16} = 1114112$). Nu toate aceste coduri sunt definite în prezent, codurile încă nedefinite putând fi definite în versiuni următoare ale standardului.

Codurile unicode sunt numere de la 0 la $2^{20} + 2^{16} - 1$. Codurile de la 0 la 127 corespund aceloraşi caractere ca şi în codificarea ASCII.

Reprezentarea codurilor unicode ca şiruri de octeţi poate fi făcută în mai multe moduri. Cele mai răspândite codificări sunt:

- *UTF-8* este o codificare de lungime variabilă, între 1 şi 4 octeţi pentru un caracter;
- *UTF-16*, *UTF-16LE*, *UTF-16BE* sunt codificări de lungime variabilă, 2 sau 4 octeţi pentru un caracter;
- *UTF-32*, *UTF-32LE*, *UTF-32BE* sunt codificări de lungime fixă, de 4 octeţi pentru fiecare caracter.

Caracter	Cod *unicode* (hexa)	Cod *unicode* (zecimal)	UTF-8 (hexa)
Ă	102	258	C4 82
ă	103	259	C4 83
Â	C2	194	C3 82
â	E2	226	C3 A2
Î	CE	206	C3 8E
î	EE	238	C3 AE
Ș	218	536	C8 98
ș	219	537	C8 99
Ț	21A	538	C8 9A
ț	21B	539	C8 9B
Ş	15E	350	C5 9E
ş	15F	351	C5 9F
Ţ	162	354	C5 A2
ţ	163	355	C5 A3

Tabelul 7.2: Caracterele cu diacritice din alfabetul limbii române şi codificările *unicode* corespunzătoare. Notă: caracterele *Ş*, *ş*, *Ţ* şi *ţ* au câte două forme utilizate: una cu virgulă dedesupt, cealaltă cu sedilă. Conform normelor stabilite de Academia Română, forma corectă este cea cu virgulă. Codificarea formei cu virgulă a fost standardizată mai recent, motiv pentru care multe documente utilizează încă forma cu sedilă.

7.2.3.1. Codificarea UTF-8

Corespondenţa de la codul caracterului la şirul de octeţi este dată în tabelul 7.3.

Valorile lui c (în baza 16)	reprezentarea UTF-8 (în baza 2)
0–7F	$0c_7c_6c_5c_4c_3c_2c_1c_0$
80–7FF	$110c_{10}c_9c_8c_7c_6$ $10c_5c_4c_3c_2c_1c_0$
800–FFFF	$1110c_{15}c_{14}c_{13}c_{12}$ $10c_{11}c_{10}c_9c_8c_7c_6$ $10c_5c_4c_3c_2c_1c_0$
10000–1FFFFF	$11110c_{20}c_{19}c_{18}$ $10c_{17}c_{16}c_{15}c_{14}c_{13}c_{12}$ $10c_{11}c_{10}c_9c_8c_7c_6$ $10c_5c_4c_3c_2c_1c_0$

Tabelul 7.3: Codificarea UTF-8. c reprezintă codul *unicode* al caracterului; $c_{20} \dots c_0$ reprezintă cifrele reprezentării binare a lui c, cu c_{20} reprezentând cifra cea mai semnificativă şi c_0 cea mai puţin semnificativă. Codificarea există doar pentru $0 \leq c < 2^{21}$.

De remarcat că schema pentru coduri mari (de exemplu, schema pentru c între 80_{16} şi $7FF_{16}$) poate fi principial aplicată şi la coduri mai mici (de exemplu, pentru $c = 41_{16}$, rezultând doi octeţi, $C1_{16}$ urmat de 81_{16}). O astfel de codificare este însă interzisă de standard pentru a asigura unicitatea codificării UTF-8.

Codificarea UTF-8 permite recuperarea sincronismului (dacă receptorul pierde câţiva octeţi poate regăsi unde începe un caracter nou), deoarece fiecare caracter nou începe cu un octet cuprins între 0 şi 127 sau între 192 şi 255, iar ceilalţi octeţ din codificarea unui caracter sunt cuprinşi între 128 şi 191. O altă proprietate este că lungimea codificării UTF-8 a unui caracter poate fi determinată după citirea primului octet.

7.2.3.2. Codificările UTF-16

Codificarea UTF-16 este descrisă în două etape: într-o primă etapă, codul *unicode* este transformat într-unul sau două numere de câte 16 biţi, iar în a doua etapă fiecare astfel de număr este scris ca 2 octeţi consecutivi.

Caracterele cu codul *unicode* între 0 şi $D7FF_{16}$ sau între $E000_{16}$ şi $FFFF_{16}$ se scriu ca un singur întreg pe 16 biţi.

Caracterele cu codul *unicode* între 10000_{16} şi $10FFFF_{16}$ se scriu ca doi întregi de câte 16 biţi astfel: Mai întâi, din codul *unicode* se scade 10000_{16}, rezultând o valoare între 0 şi $FFFFF_{16}$ (20 biţi). Primul întreg de 16 biţi se formează punând cifrele 110110 urmate de primii 10 din cei 20 de biţi. Al doilea întreg se formează punând cifrele 110111 urmate de ultimii 10 din cei 20 de biţi. De exemplu, codul *unicode* 10302_{16} se scrie ca doi întregi astfel: $D83C_{16}$ $DF02_{16}$.

Într-o a doua etapă este definită scrierea fiecărui întreg de 16 biţi ca un şir de doi octeţi. Există două modalităţi de a reprezenta fiecare astfel de întreg, începând de la octetul mai semnificativ (de rang mai mare) sau începând de la octetul mai puţin semnificativ. Pentru a reflecta aceste variante diferite de alegere, există trei codificări distincte numite generic *UTF-16*:

- *UTF-16LE*: Primul octet este cel mai puţin semnificativ (*little endian*);
- *UTF-16BE*: Primul octet este cel mai semnificativ (*big endian*);
- *UTF-16*: Ordinea octeţilor poate fi fie *big endian*, fie *little endian*, la alegerea emiţătorului. Primul caracter codificat trebuie să fie caracterul cu codul FEFF_{16} (definit iniţial ca fiind un caracter de control ce interzice ruperea în rânduri în acel punct, dar este utilizat în prezent doar ca marcaj pentru identificarea ordinii octeţilor). Ordinea octeţilor este dedusă de receptor prin examinarea primilor doi octeţi: dacă aceştia sunt FE_{16} urmat de FF_{16}, înseamnă că ordinea octeţilor este *big endian*; dacă este FF_{16} urmat de FE_{16}, înseamnă că ordinea octeţilor este *little endian*.

7.2.3.3. Codificările UTF-32

Codificarea UTF-32 constă în codificarea fiecărui caracter ca un întreg pe 32 de biţi, reprezentat la rândul lui ca un şir de 4 octeţi. Ca şi în cazul codificărilor *UTF-16*, există trei codificări *UTF-32*:

- *UTF-32LE*: Primul octet este cel mai puţin semnificativ (*little endian*);
- *UTF-32BE*: Primul octet este cel mai semnificativ (*big endian*);
- *UTF-32*: Ordinea octeţilor poate fi fie *big endian*, fie *little endian*, la alegerea emiţătorului. Primul caracter codificat trebuie să fie caracterul cu codul $FEFF_{16}$. Ordinea octeţilor este dedusă de receptor prin examinarea primilor patru octeţi: dacă aceştia sunt 0, 0, FE_{16}, FF_{16}, înseamnă că ordinea octeţilor este *big endian*; dacă este FF_{16}, FE_{16}, 0, 0, înseamnă că ordinea octeţilor este *little endian*.

7.3. Reprezentarea datei şi orei

Determinarea datei şi orei producerii unui eveniment, precum şi memorarea sau transmiterea acestora, sunt operaţii frecvente într-o reţea de calculatoare.

Problema reprezentării datei şi orei este mult mai dificilă decât pare la prima vedere. Din acest motiv, vom începe prin a studia ce se poate înţelege

prin „ora curentă“, iar apoi vom vedea ce scheme de reprezentare ale datei şi orei există şi ce avantaje şi dezavantaje aduce fiecare dintre ele.

7.3.1. Măsurarea timpului

Există două metode utilizate pentru indicarea timpului curent (datei şi orei curente):

- pe baza unor fenomene astronomice, anume alternanţa zi-noapte (în termeni astronomici, *ziua solară mijlocie*), alternanţa anotimpurilor (*anul tropic*) şi, eventual, fazele lunii (*luna sinodică*);
- pe baza unui fenomen fizic repetabil, de exemplu oscilaţia unui pendul sau vibraţia unui cristal de cuarţ.

Prima variantă este de interes practic imediat pentru sincronizarea activităţilor umane. Are însă complicaţii inerente legate de următoarele fapte:

- alternanţa zi-noapte nu este simultană pe tot Pământul ci este decalată pe longitudine;
- anul, luna şi ziua sunt incomensurabile (rapoartele duratelor lor sunt numere iraţionale);
- anul, luna şi ziua nu au durate constante (fenomenele corespunzătoare nu sunt perfect periodice) şi nici măcar previzibile exact (în special rotaţia Pământului are neuniformităţi imprevizibile datorate redistribuirii masei în interiorul Pământului).

A doua variantă măsoară direct timpul ca mărime fizică şi oferă avantaje atunci când avem de determinat ordinea cronologică a unor evenimente sau de calculat duratele de timp dintre ele. Timpul (fizic) a ajuns să poată fi definit independentă de mişcarea Pământului doar după dezvoltarea, începând cu anii 1950, a ceasurilor atomice, mai precise decât mişcările Pământului. Măsurarea timpului se face pe baza *secundei* definite în Sistemul Internaţional de unităţi (SI) ca *9192631770 de perioade ale oscilaţiei corespunzătoare tranziţiei între cele două nivele hiperfine ale stării fundamentale a atomului de cesiu 133*.

Ca urmare a acestor complicaţii, există mai multe standarde de măsurare şi reprezentare a timpului:

Timpul atomic internaţional (TAI) este dat de numărul de secunde SI scurse de la un anumit moment ales ca reper. Secundele TAI se grupează în minute, ore, zile, etc.

Timpul universal UT1 este de fapt măsura unui anumit unghi, legat de rotaţia Pământului, exprimată în unităţi de timp (24 h în loc de 360°). (Unghiul respectiv este unghiul orar, pentru un observator aflat

pe meridianul 0°, al soarelui mijlociu.) Curge neuniform datorită neuniformităţii mişcării de rotaţie a Pământului; după media ultimilor câţiva ani, 24 h UT1 este aproximativ 86400,002 s SI.

Timpul universal coordonat (UTC) este bazat pe secunda SI, dar gruparea secundelor în zile este modificată pentru a menţine diferenţa dintre UT1 şi UTC la sub o secundă.

Astfel, o zi UTC normală are 24 ore a 60 minute a 60 secunde SI fiecare, adică 86400 s. Dacă UT1−UTC se apropie de −1 s, se adaugă o secundă de corecţie (engl. *leap second*) la o zi, astfel încât acea zi UTC are 86401 s, proces aproape echivalent cu a muta UTC cu o secundă înapoi. În acest scop, ultimul minut al zilei are 61 de secunde în loc de 60, după ora 23:59:59 urmează, la o secundă, 23:59:60 şi abia după încă o secundă ora 0:00:00 a zilei următoare. Dacă UT1−UTC se apropie de 1 s, se elimină o secundă din ultimul minut al unei zile, astfel că la o secundă după 23:59:58 urmează ora 0:00:00 a zilei următoare. Din anul 1972 (de la introducerea UTC în forma actuală) până în 2008 au fost adăugate 23 de secunde de corecţie şi nu a fost eliminată nici una. A 24-a secundă de corecţie se va adăuga la sfârşitul anului 2008, astfel încât ziua de 31 decembrie 2008 va avea 86401 secunde. Datorită unei diferenţe iniţiale de 10 s între TAI şi UTC, diferenţa TAI−UTC este în prezent de 33 s.

Timpul legal în fiecare ţară este definit fie pe baza UT1, fie pe baza UTC (diferenţa este neglijabilă pentru uzul practic), ca fiind UTC (sau UT1) plus sau minus un anumit număr de ore şi uneori şi fracţiuni de oră (exemplu, India are ora legală UTC+5h30min).

În ţările în care există oră de vară, la trecerea de la ora de iarnă la cea de vară şi invers, diferenţa dintre ora legală şi UTC creşte, respectiv scade, cu o oră (de notat că UTC nu are oră de vară). De exemplu, ora legală în România este UTC+2 h în timpul iernii (din ultima duminică din octombrie până în ultima duminică din martie) şi UTC+3 h în timpul verii.

Ora suplimentară introdusă la trecerea de la ora de vară la cea de iarnă nu are notaţie distinctă, de tipul secundelor de corecţie din UTC. Ca urmare, la trecerea de la ora de vară la ora de iarnă, există perechi de momente de timp care sunt notate la fel şi ca urmare ora legală este ambiguă. Exemplu: dacă prin trecerea de la ora de vară la cea de iarnă ora 4:00:00 devine 3:00:00, atunci notaţia 3:30:00 poate corespunde la două momente de timp, ora de vară 3:30:00 (la 30 min înaintea schimbării orei) şi ora de iarnă 3:30:00 (la 30 min după schimbarea orei).

Pentru gruparea zilelor în unităţi mai mari, în special în ani, sunt utilizate mai multe sisteme (calendare):

Calendarul gregorian , introdus în anul 1582 şi în vigoare în România din anul 1924, este calendarul actual. Anii bisecţi (de 366 de zile) sunt anii cu numărul anului divizibil cu 4, cu excepţia celor divizibili cu 100 fără a fi divizibili cu 400. Ani bisecţi sunt 1600, 2000, 2400 etc; ani nebisecţi divizibili cu 100 sunt 1700, 1800, 1900, 2100, 2200 etc. Durata medie a anului gregorian este 365,2425 zile, ceva mai lung decât anul tropic de aproximativ 265,2422 zile.

Calendarul iulian , predecesorul calendarului gregorian, introdus în anul 45 î.e.n. şi având regula mai simplă cum că sunt bisecţi toţi anii cu număr divizibil cu 4. Este utilizat adesea de istorici pentru a data şi evenimente dinainte de anul 45 î.e.n., caz în care el este numit *calendar iulian proleptic*. Cu o durată medie a anului de 365,25 zile, calendarul iulian rămâne în urmă cu 1 zi la aproximativ 128 de ani.

Ziua iuliană este un simplu număr ce arată numărul de zile scurse de la o dată de referinţă. Acest sistem este utilizat frecvent în astronomie deoarece permite uşor calculul duratelor dintre două date; din acelaşi motiv reprezintă o schemă potrivită pentru reprezentarea datei în calculator. Există în două variante. Prima, JD (*julian day*), are ca referinţa data de 1 ianuarie 4713 î.e.n. conform calendarului iulian proleptic, la amiaza UT1. Momentul respectiv este JD 0,0, miezul nopţii următoare este JD 0,5, etc. Cealaltă, MJD (*modified julian day*), are ca referinţă 17 noiembrie 1858 ora 0, adică este JD$-$2400000,5 .

7.3.2. Obiectivele în alegerea reprezentării timpului în calculator

De obicei, operaţiile ce trebuiesc efectuate asupra reprezentării timpului sunt:

1. Citirea sau scrierea timpului ca oră legală conform calendarului gregorian în formatul obişnuit, precum şi efectuarea de operaţii aritmetice de genul adunării sau scăderii unei durate formale (exemplu: mâine la aceeaşi oră; aceasta înseamnă în mod obişnuit peste 24 de ore, dar poate însemna peste 23 sau 25 de ore dacă intervine trecerea de la ora de iarnă la cea de vară sau invers).
2. determinarea ordinii cronologice a două momente de timp;
3. determinarea exactă, ca timp fizic, a duratei între două momente de timp,

4. pentru aplicaţii speciale, citirea sau afişarea timpului în alte formate (timpul legal al altui fus orar, UTC, TAI, JD, etc).

Punctul 1 este cerut de toate sistemele. Punctul 2 este important pentru foarte multe aplicaţii şi rezolvarea lui corectă interzice mutarea ceasului înapoi. Punctul 3 este important în aplicaţiile în timp real; de asemenea, funcţionarea ceasului sistem presupune, în mod repetat, adunarea unei durate de timp la un moment de timp.

Reprezentarea directă a orei legale, sub forma an, lună, zi, oră, minut, secundă, fracţiuni de secundă, rezolvă simplu punctul 1. Ea ridică însă probleme la punctul 2 dacă sunt implicate calculatoare aflate pe fusuri orare distincte sau dacă se efectuează operaţii în intervalul de o oră în jurul trecerii de la ora de vară la cea de iarnă; pentru tratarea corectă a acestor cazuri este necesar să se ştie, despre fiecare oră, pe ce fus orar este considerată şi care sunt regulile privind ora de vară. Punctul 3 ridică, pe lângă problemele comune cu cele legate de punctul 2, complicaţii legate de saltul cu o oră înainte la trecerea de la ora de iarnă la ora de vară şi calculele legate de calendar; de asemenea, pentru calcule exacte ale duratelor, sunt necesare informaţii cu privire la secundele de corecţie.

Reprezentarea orei UTC permite determinarea ordinii cronologice şi a duratelor fără a necesita date despre fusurile orare sau regulile privind ora de vară, în schimb aceste date sunt necesare la conversia între reprezentarea UTC şi timpul legal.

Reprezentarea TAI ca număr de unităţi de timp scurse de la un anumit moment fixat rezolvă extrem de simplu punctele 2 şi 3 în schimb mută dificultăţile la rezolvarea punctului 1.

7.3.3. Formate utilizate în practică

Deoarece într-o reţea pot fi prezente calculatoare situate pe fusuri orare distincte, aproape orice format util în reţea fie transmite direct ora UTC sau TAI, fie transmite suficientă informaţie pentru ca receptorul să poată calcula uşor ora UTC.

7.3.3.1. Formatul utilizat de poşta electronică

Pentru poşta electronică (§ 11.1), reprezentarea datei se face ca text şi conţine, în ordine:

- opţional ziua din săptămână, ca prescurtare de 3 litere din limba engleză),
- ziua, ca număr între 1 şi 31,
- luna, ca şir de trei litere, prescurtare din engleză,

- anul, ca şir de 4 cifre,
- ora, totdeauna ca 2 cifre, între 00 şi 23,
- minutul, ca două cifre, între 00 şi 59,
- opţional, secunda, ca două cifre între 00 şi 60,
- diferenţa dintre ora legală conform căreia a fost scrisă data şi ora UTC; aceasta este dată ca 4 cifre, 2 pentru numărul de ore şi 2 pentru numărul de minute, cele patru cifre fiind precedate de semnul + sau −. Componentele datei sunt separate printr-un amestec de virgule, spaţii şi caractere *două puncte*.

De exemplu, data:

```
Thu, 25 Oct 2007, 17:22:19 +0300
```

înseamnă că la momentul scrierii mesajului ora locală a expeditorului era joi, 25 octombrie 2007, ora 17:22:19 şi că ora respectivă este cu 3 ore în avans faţă de UTC. Ca urmare, ora UTC la acel moment era 14:22:19.

Data considerată în acest exemplu este plauzibilă conform orei legale a României, în 25 octombrie 2007 fiind încă în vigoare ora de vară care este cu 3 ore în avans faţă de UTC.

Orele astfel specificate sunt uşor de comparat şi nu există ambiguităţi legate de trecerea de la ora de vară la cea de iarnă. De exemplu, un mesaj trimis înainte de trecerea la ora de iarnă ar fi datat

```
Sun, 28 Oct 2007, 03:40 +0300
```

urmat la jumătate de oră, după trecerea la ora de iarnă, de

```
Sun, 28 Oct 2007, 03:10 +0200
```

7.3.3.2. ISO-8601 şi RFC-3339

ISO-8601 este o standardizare a modului de scriere ca text a datei şi orei. Standardul fiind foarte complex, a apărut RFC-3339 care cuprinde cazurile mai utile şi mai frecvent folosite din ISO-8601.

RFC-3339 prevede reprezentarea datelor astfel:

- anul, ca patru cifre (nu sunt admise prescurtări de genul 07 pentru 2007),
- luna, ca 2 cifre (01 pentru ianuarie, 12 pentru decembrie),
- ziua, ca 2 cifre (de la 01 până la 31 sau mai puţin, în funcţie de lună).

Cele trei componente sunt separate prin liniuţe (ISO-8601 permite şi alipirea lor):

```
2007-10-28
```

Ora se reprezintă prin 2 cifre pentru oră (00–23), 2 cifre pentru minut (00–59), două cifre pentru secundă (00–60, în funcţie şi de prezenţa unei secunde de corecţie), eventual fracţiunile de secundă şi eventual specificarea fusului orar. Ora, minutul şi secunda sunt separate prin *două puncte*, fracţiunile de secundă se separă de câmpul pentru secunde prin *punct*, iar specificarea fusului orar se face printr-un semn plus sau minus urmat de două cifre pentru numărul de ore diferenţă urmat de caracterul *două puncte* şi încă două cifre pentru numărul de minute diferenţă. Exemplu:

```
21:12:58.342+02:00
```

reprezintă acelaşi moment de timp, dar pe alt fus orar, cu

```
14:12:58.342-05:00
```

Data şi ora se specifică împreună punând litera `T` între ele:

```
2007-10-28T14:12:58.342-05:00
```

7.3.3.3. Timpul POSIX

În sistemele de tip UNIX (conforme standardului POSIX), reprezentarea timpului este făcută printr-un număr întreg considerat de obicei că reprezintă numărul de secunde scurse de la 1 ianuarie 1970 ora 0:00 UTC. Ora UTC în formatul obişnuit se obţine grupând secundele în minute, ore, zile, luni şi ani conform regulilor obişnuite. De fapt, numărul dat ca dată nu este exact numărul de secunde scurse de la 1 ianuarie 1970, ci diferă de acesta prin numărul de secunde de corecţie adăugate pentru menţinerea în sincronism a UTC cu rotaţia Pământului. De aceea, „timpul unix“ are salturi înapoi de câte o secundă la fiecare introducere a unei astfel de secunde de corecţie. Timpul POSIX este reprezentat în mod obişnuit ca întreg cu semn pe 32 de biţi, şi ca urmare valoarea cea mai mare ce poate fi reprezentată corespunde datei de 19 ianuarie 2038, ora 3:14:07 UTC.

7.3.3.4. TAI 64

TAI 64 este un standard ce presupune reprezentarea timpului ca număr de secunde, incluzând secundele de corecţie. Numărul de secunde este reprezentat pe 63 de biţi (plus un bit rezervat), cu valoarea 2^{62} corespunzând datei de 1 ianuarie 1970 ora 0 TAI. Intervalul de timp reprezentabil este imens, de ordinul a 10^{11} ani.

Pentru aplicaţii ce au nevoie de rezoluţie mai bună de o secundă, standardul prevede încă două câmpuri de câte 32 de biţi (total 128 de biţi), reprezentând respectiv numărul de nanosecunde şi de attosecunde (valori între 0 şi $10^9 - 1$).

7.4. Recodificări

Este necesar uneori să codificăm un şir mai mult sau mai puţin arbitrar de octeţi sub forma unui şir de caractere supus unor restricţii. Astfel de situaţii apar:

- La trimiterea fişierelor ataşate la mesajele de poştă electronică, întregul mesaj, inclusiv partea ce cuprinde fişierele ataşate, trebuie să îndeplinească anumite restricţii, între altele, să nu conţină caractere ASCII de control cu excepţia perechilor *carriage return–line feed* de la finalul fiecărui rând, să nu aibă rânduri prea lungi, etc. Pe de altă parte, fişierele ataşate pot fi fişiere binare cu conţinut arbitrar.
- La stocarea în fişiere text a unor informaţii reprezentate natural ca şir arbitrar de octeţi, de exemplu la stocarea în fişiere text a unor chei de criptare, semnături electronice, etc.
- În limbaje de programare, la scrierea în şirurile de caractere a unor caractere cu rol special, ca de exemplu a ghilimelelor (care în mod normal sunt interpretate ca terminatorul şirului de caractere).

În astfel de situaţii, este necesar să se codifice un şir arbitrar de octeţi (fiecare octet poate lua orice valori între 0 şi 255) pe un alfabet constând în litere, cifre şi câteva simboluri speciale. Deoarece numărul de simboluri disponibile este mai mic decât numărul de valori posibile ale octeţilor, un octet nu se va putea codifica numai pe un singur caracter.

7.4.1. Codificarea hexazecimală

Codificarea hexazecimală prevede ca fiecare octet să se reprezinte pe două caractere, fiecare caracter putând fi din mulţimea cuprinzând cifrele zecimale (0–9) şi literele *A–F*.

Exemplu: şirul de octeţi 120, 0, 23, 45, 20 se scrie: `7800172D14`.

Uneori, în şirul de cifre hexa se inserează spaţii sau caractere *newline* pentru lizibilitate sau pentru evitarea rândurilor foarte lungi.

Deoarece un caracter se reprezintă de obicei pe un octet, prin această recodificare rezultă o dublare a lungimii şirului.

7.4.2. Codificarea în baza 64

Codificarea *în baza 64* codifică un şir de 3 octeţi cu valori arbitrare ca un şir de 4 caractere din mulţimea cuprinzând literele mari, literele mici, cifrele şi caracterele `+`, `/` şi `=`.

Codificarea se face în modul următor:

1. Şirul iniţial de octeţi se completează la un multiplu de 3 octeţi prin adăugarea a 0, 1 sau 2 octeţi cu valoarea 0.
2. Se formează un şir de 24 de biţi punând unul după altul cei câte 8 biţi din fiecare octet; din fiecare octet se începe cu bitul cel mai semnificativ.
3. Şirul de 24 de biţi se împarte în 4 grupuri de câte 6 biţi.
4. Fiecare şir de 6 biţi se consideră ca fiind un număr cuprins între 0 şi 63, considerând primul bit din şir ca fiind cel mai semnificativ.
5. Fiecare număr obţinut la pasul anterior se reprezintă printr-un caracter. Cele 64 de valori posibile, de la 0 la 63, se reprezintă ca: litere mari (0→`A`, 25→`Z`), litere mici (26→`a`, 51→`z`), cifre (52→`0`, 61→`9`) şi caracterele `+` şi `/` (62→`+`, 63→`/`). Dacă o valoare 0 provine din biţi 0 proveniţi integral dintr-un octet completat la pasul 1, în loc de `A` se scrie `=`.

De exemplu, şirul de octeţi 120, 0, 23, 45, 20 devine şirul de biţi

01111000 00000000 00010111 00101101 00010100 00000000

ultimul octet fiind adăugat la pasul 1. Şirul de biţi se regrupează

011110 000000 000000 010111 001011 010001 010000 000000

rezultând şirul de numere „în baza 64“: 30, 0, 0, 23, 11, 17, 16, 0, care se codifică `eAAXLRQ=`

7.4.3. Codificări bazate pe secvenţe de evitare

Recodificările prin secvenţe de evitare sunt utilizate în situaţia în care majoritatea octeţilor sau caracterelor din textul de recodificat sunt codurile unor caractere ce se regăsesc în alfabetul în care se face recodificarea. În această situaţie, este favorabil ca, pe cât posibil, octeţii sau caracterele din textul de recodificat să fie codificate prin ele însele, în special pentru ca textul să poată fi înţeles (parţial, cel puţin) de către un utilizator uman direct în forma recodificată.

Recodificarea se face astfel. Se distinge un caracter în alfabetul destinaţie, caracter ce este denumit *caracter de evitare* (enlg. *escape character*).

- Orice caracter din alfabetul sursă care se regăseşte în alfabetul destinaţie şi este diferit de caracterul de evitare se recodifică ca el însuşi.

- Caracterul de evitare (dacă face parte din alfabetul sursă), precum şi caracterele din alfabetul sursă ce nu se găsesc în alfabetul destinaţie, se codifică ca secvenţe de caractere ce încep cu un caracter de evitare.

  Exemple:

- Pentru poşta electronică, un caracter ce nu pot fi transmise direct este codificat ca o secvenţă de trei caractere, un caracter *egal* (`=`) şi două cifre hexa reprezentând codul caracterului de transmis. Această recodificare se numeşte *quouted printables*. Caracterele ASCII imprimabile, cu excepţia caracterului *egal*, se transmit direct (fără recodificare). Ca urmare, un text în care caracterele ce trebuie recodificate sunt rare poate fi înţeles de către un utilizator uman fără prea mari dificultăţi. Ca exemplu, fraza precedentă se scrie (presupunând o codificare ISO-8859-2 pentru caractere şi apoi o recodificare *quouted printables*):

  ```
  Ca urmare, un text =EEn care caracterele ce trebuie
  recodificate sunt rare poate fi =EEn=FEeles de c=E3tre
  un utilizator uman f=E3r=E3 prea mari dificult=E3=FEi.
  ```

- În *URL*-uri, caracterele ce au rol sintactic (*spaţiu*, `/`, etc) se codifică prin caracterul *procent* (`%`) urmat de două cifre hexa reprezentând valoarea caracterului respectiv. De notat că aceste coduri sunt în cadrul codificării UTF-8; ca urmare, o pagină cu numele *şir* ar avea un URL de forma

  ```
  http://example.com/%C8%98ir
  ```

- În limbajul C şi în alte limbaje derivate din el, ghilimelele (`"`) servesc la delimitarea şirurilor de caractere. Dacă se doreşte introducerea unor astfel de caractere într-un şir, sau a caracterelor ASCII de control, se introduc secvenţe *escape* cum ar fi: `\"` pentru ghilimele (`"`), `\\` pentru *backslash* (`\`), `\n` pentru *newline* (caracterul ASCII cu codul 10).
- În HTML, caracterele *unicode* sunt scrise prin secvenţe `&#`*cod*`;`. De exemplu, litera *ţ* se scrie `&#539;`.