ARHITECTURA CALCULATOARELOR 2003/004
CURSUL 2

1.2.2 Organizarea memoriei principale (continuare)

Memoria este alcatuită dintr-un numar de celule (sau locații), fiecare putând stoca o parte din informație. Fiecare celulă are un numar, numit adresă, la care programele se pot referi. Dacă o memorie are n locații, ele vor avea adrese de la 0 la n-1. Toate locațiile dintr-o memorie contin acelaşi număr de biți. Dacă o locație conține k biți, atunci ea poate memora 2^k combinații diferite. Figura următoare arată trei moduri diferite de organizare pentru o memorie de 96 biți. Notați că locațiile alăturate au adrese consecutive. Calculatoarele care folosesc sisteme de numere binare exprimă adresele memoriei ca numerele binare. Dacă o adresă are m biți, numărul maxim de locații adresabile este de 2^m. De exemplu, o adresă care face referință la memoria din figura 18a are nevoie de cel putin 4 biți pentru a exprima toate numerele de la 0 la11. O adresă cu 3 biți este suficientă pentru situațiile din figurile 18b respectiv 18c. Numărul de biți din adresă determină numărul maxim de locații adresabile dirtect în memorie şi este independent de numărul de biți al fiecărei locații.

O memorie cu 2^{12} locații a câte 8 biți fiecare şi o memorie cu 12^{12} locații a câte 64 biți fiecare ambele necesită adrese de 12 biți. Numărul de biți pe locații ale unor calculatoare care au existat pe piață în timp sunt listate în figura 1.9.

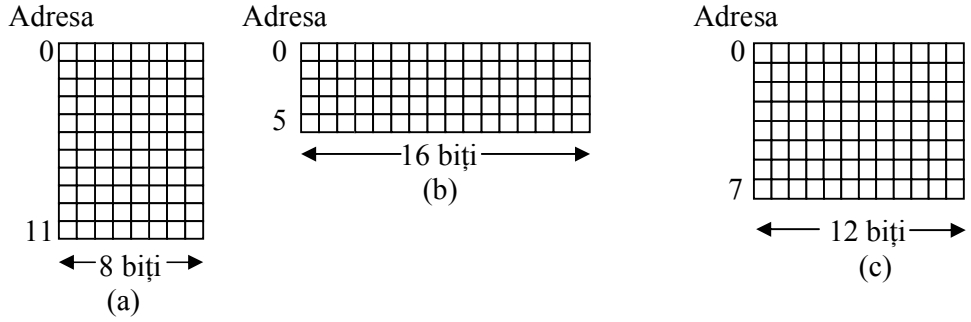


Figura 1.8 Trei variante de a organiza o memorie de 96 de biți

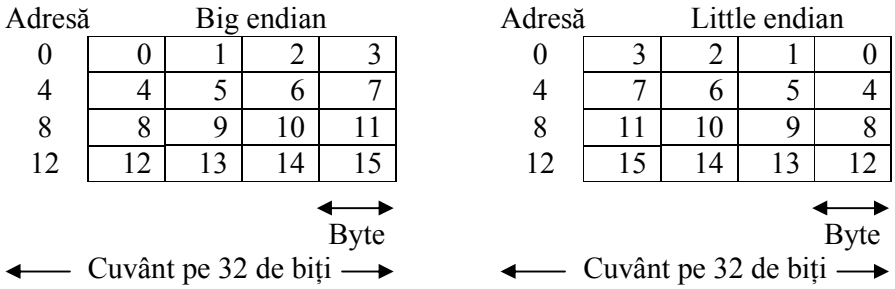
Calculatoare	Biți/Celulă
Burroughs B1700	1
IBM PC	8
DEC PDP-8	12
IBM 1130	16
DEC PDP-15	18
XDS 940	24
Electrologica X8	27
XDS Sigma 9	32
Honeywell 6180	36
CDC 3600	48
CDC Cyber	60

Figura 1.9 Numărul de biți pe celulă la unele calculatoare istorice

Semnificatia celulei este aceea că este cea mai mică unitate adresabilă. În ultimii ani, fabricile de calculatoare au standardizat locația la 8 biți şi au denumit-o byte. Bytes sunt grupați în words. Un calculator cu 32 bit/word are 4 bytes/ word, iar un calculator cu 64 bit/word are 8 bytes/word. Semnificația unui cuvant (word) este aceea că majoritatea instrucțiunilor operează pe un numar întreg de cuvinte (words), de exemplu grupând două cuvinte împreună. Deci o maşină pe 32 biți va avea regiştrii de 32 biți şi instrucțiuni pentru manipularea cuvintelor de 32 biți, iar una pe 64 biți va avea regiştrii pe 64 biți şi instrucțiuni pentru mutarea, adunarea, scăderea şi alte manipulari pe cuvinte de 64 biți.

1.2.3 Octet ordonat

Un octet într-un cuvânt poate fi numărat de la stânga la dreapta sau de la dreapta la stânga. La început acest lucru nu pare a avea importanță, dar vom vedea că el are implicații majore. Figura 1.10a prezintă o parte a memoriei unui calculator pe 32 de biți ai cărui octeți sunt numărați de la stânga la dreapta, cum ar fi SPARC sau IBM. Figura 10b arată reprezentarea analoagă la nivel de bit folosind numărarea de la dreapta la stânga, ca la familia Intel. Sistemele în care număratoarea începe cu 'big', adică cu ponderea mare, este denumit generic big endian, iar celelalte little endian. Se face trimitere la gluma din Gulliver legată de disputa "la care cap trebuie spart oul?"!



(a) Memorie tip big endian (b) Memorie tip little endian

Figura 1.10 Variante de ordonare a octeților

Este important de reținut că şi în varianta big endian şi în varianta little endian valoarea întreagă 6 se reprezintă pe 32 biți prin 110 în cei mai din dreapta 3 biți ai cuvantului şi 0 pe celelălte poziții din stânga. În varianta big endian biți 110 sunt în byte-ul 3 (sau 7 sau 11 etc.) în timp ce la little endian aceeaşi biți sunt în byte-ul 0 (sau 4 sau 8 etc). În ambele cazuri cuvântul care conține acest întreg are adresa 0.

În cazul în carea calculatoarele ar fi stocat numai numere întregi, nu ar fi fost nici o problemă. Totuşi, multe aplicații necesită un amestec de întregi, şiruri de caractere şi alte tipuri de date. Considerați, de exemplu, o simplă înregistrare pentru o persoană, care constă într-un şir de caractere (numele angajatului) şi două numere (vârsta şi numărul departamentului). Şirul este terminat cu unul sau mai mulți octeți 0, pentru a umple cuvâtul. Reprezentarea big endian este arătată în figura 1.11a, reprezentarea little endian este arătată în 1.11b pentru Jim Smith, 21 ani, departamentul 260 (1x256 +4=260).

Ambele reprezentări sunt compatibile cu ele însele. Problemele încep atunci când una dintre maşini încearcă să trimită înregistrarea celeilalte printr-o rețea. Să presupunem că big endian

rimite înregistrarea little endian, octet după octet, începând cu bitul 0 şi terminând cu bitul 19 (vom fi optimişti şi vom presupune că biţii octeţilor nu s-au inversat prin transmisie, deoarece avem deja destule probleme). Astfel, octetul 0 al big endian-ului trece în memoria little endian-ului la octetul 0, şi tot asa, ca în figura 1.11c).

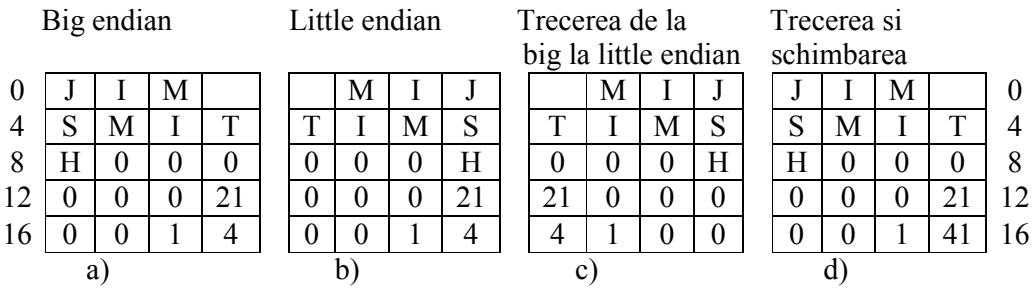


Figura 1.11 Exemplu de reprezentare pentru big şi little endian

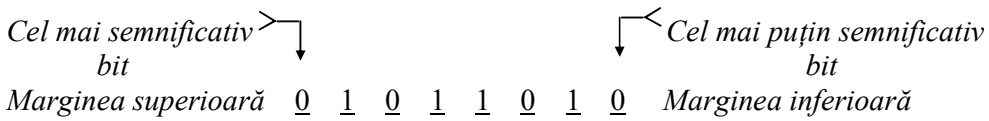
Când little endian încearcă să tipărească un nume, totul funcţionează bine, dar vârsta iese ca $21x2^{24}$, iar departamentul este şi el alterat. Această situaţie apare din cauză că transmisia a inversat ordinea caracterelor dintr-un cuvânt, cum trebuia, dar a schimbat şi octeţii dintr-un întreg, ceea n-ar fi trebuit.

O soluţie evidentă ar fi să inversăm prin software octeţii dintr-un cuvânt după efectuarea transferului. Aceasta conduce la situaţia din figura 1.11d, care prelucrează bine cei doi întregi, dar converteşte şirul în "MIJTIMS", lăsându-l pe H să atârne în aer. Această inversare se produce deoarece când îl citeşte, calculatotorul citeşte prima dată octetul 0 (un spaţiu), apoi bitul 1(M), şi tot aşa.

Soluţia nu este simplă. Un mod care funcţionează, dar este ineficient, este să includem un antet înaintea fiecărui articol, care să spună ce tip de date urmează (şir de caractere, întregi sau altele), şi cât este de lung. Aceasta permite receptorului să execute numai conversiile necesare. În orice caz, trebuie să fie clar că lipsa unui standard pentru ordonarea octeţilor este o problemă majoră în cazul schimbului de date între maşini diferite.

În cele ce urmează, vom considera biţii dintr-o celulă de memorie ca fiind aranjaţi pe un rând. Vom denumi un capăt al acestui rând **marginea superioară** şi celălalt capăt **marginea inferioară**. Cu toate că într-un calculator nu există stânga sau dreapta, ne vom imagina biţii aranjaţi într-un rând orientat de la stânga la dreapta cu marginea superioară plasată la stânga. Bitul de la acest capăt este adesea denumit **cel mai semnificant bit**; similar, bitul de la celălalt capăt este denumit bitul de la marginea inferioară sau **cel mai puţin semnificativ bit.**

Figura 1.12 Organizarea unei celule de memorie cu dimensiune de un octet



Procedând astfel, putem reprezenta conţinutul unei celule cu dimensiune de un octet ca în figura 1.12.

O consecinţă inportantă a ordonării celulelor în memoria principală, precum şi a biţilor în interiorul fiecărei celule este aceea că toţi biţii din principală a unui calculator sunt ordonaţi într-un unic şir lung. Astfel, părţi din acest şir pot fi utilizate pentru stocarea de şiruri de biţi care pot fi mai mari decât lungimea unei singure celule. În particular, dacă memoria este împărţită în celule de dimensiunea unui octet, putem stoca un şir de 16 biţi utilizând pur şi simplu două celule de memorie consecutive.

1.3 Codificarea utilizată pentru stocarea informaţiilor

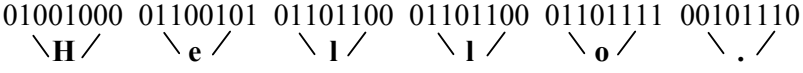
În cele ce urmează vom studia mai în amănunt tehnicile utilizate pentru reprezentarea informaţiilor sub formă de şiruri de biţi.

1.3.1 Reprezentarea simbolurilor

O procedură de reprezentare a datelor într-un calculator este aceea de a se proiecta un cod în care diferite simboluri (cum ar fi literele din alfabet sau semnele de punctuaţie) au asociate modele (şabloane) de biţi unice şi de a stoca apoi informaţia sub forma unor propoziţii codificate în memoria principală a calculatorului, respectiv pe un mediu de stocare de masă. La începutul utilizării calculatoarelor, pentru diferite echipamente au fost create şi folosite alte coduri, acest lucru având ca efect apariţia şi proliferarea problemelor de comunicaţie.

Pentru a remedia situaţia, Institutul American Naţional pentru Standarde (American National Standards Institute - ANSI) a adoptat codul **American Standard Code for Information Interchange (ASCII** pronunţat ″aschii″**),** care a devenit extrem de popular. Acest cod utilizează modele cu o lungime de şapte biţi pentru reprezentarea literelor mari şi mici ale alfabetului englez, semnelor de punctuaţie, cifrelor de la 0 la 9, precum şi a anumitor caractere de control cum ar fi trecerea la rândul următor (line feed), revenirea la marginea din stânga textului (carriage return) sau tabulator (tab). În prezent, codul ASCII este adesea extins la un format de opt biţi pe simbol prin adăugarea unui zero pe poziţia celui mai semnificativ bit în faţa fiecărui model de şapte biţi al vechiului cod. Această tehnică nu numai că produce un cod ale cărui cuvinte au dimensiunea egală cu a unei celule uzuale de memorie, dar furnizează alte 128 şabloane suplimentare (care se obţin prin plasarea vlorii 1 pe poziţia bitului cel mai semnificativ din octet), permiţând astfel reprezentarea simbolurilor excluse din codul ASCII iniţial. Din nefericire, datorită faptului că în general fabricanţii dau propriile lor interpretări acestor caractere suplimentare, adesea datele care conţin şabloane extinse sunt dificil de transferat între diferite aplicaţii. Figura 1.13 arată că, în acest sistem, şirul de biţi 01001000 01100101 01101100 01101100 01101111 00101110 reprezintă mesajul ″Hello.″.

Figura 1.13 Mesajul ″Hello.″ scris în cod ASCII



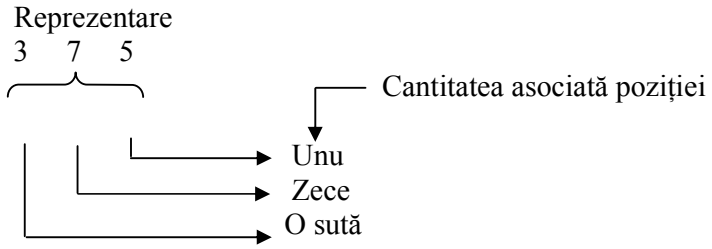
Cu toate că în prezent ASCII este cel mai utilizat cod, alte coduri mai puternice capabile să reprezinte documente scrise într-o varietate de limbaje, câştigă în popularitate. Unul dintre

acestea, Unicode, a fost dezvoltat în coooperare de către unii dintre cei mai importanţi producători de echipamente şi programe. El utilizează modele de 16 biţi pentru reprezentarea fiecărui simbol. Ca rezultat, codul Unicode conţine 65.536 şabloane diferite - suficiente pentru a permite reprezentarea celor mai utilizate ideograme chinezeşti şi japoneze. Un cod care va concura probabil cu Unicode este dezvoltat de Organizaţia Internaţională de Standarde (**International Standard Organiztion - ISO**), din care face parte şi ANSI. Utilizând modele de 32 de biţi pentru reprezentarea simbolurilor, acest cod poate reprezenta peste 17 milioane de simboluri. Rămâne de văzut care dintre coduri va câştiga cea mai mare popularitate.

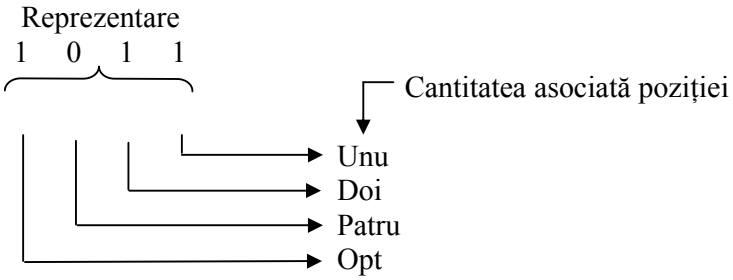
1.3.2 Reprezentarea valorilor numerice

Deşi metoda de stocare a informaţiilor sub forma unor caractere codificate este foarte utilă, ea este ineficientă atunci când informaţiile care trebuie memorate sunt de natură numerică. Pentru a înţelege de ce se întâmplă acest lucru, să presupunem că vrem să stocăm numărul 25. Dacă dorim să-l înregistrăm sub forma unor simboluri ASCII codificate cu opt biţi pe simbol, ne vor trebui în total 16 biţi. Mai mult, cel mai mare număr pe care-l putem stoca utilizând 16 biţi este 99. O abordare mult mai eficientă este stocarea valorii reprezentate în baza doi (binar).

Figura 1.14 Sistemul zecimal şi sisemul binar de numeraţie



(a) Sistemul zecimal de numeraţie



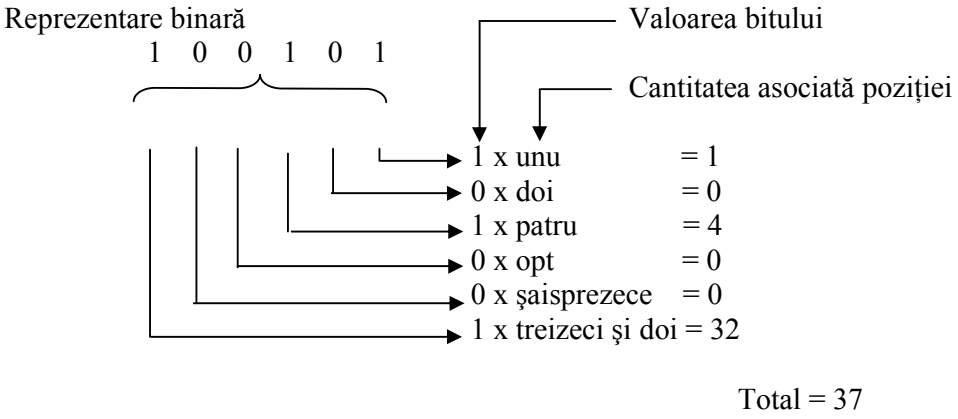
(b) Sistemul binar de numeraţie

Notaţia binară este o metodă de reprezentare a valorilor numerice folosindu-se numai cifrele 0 şi 1 în loc să se utilizeze cifrele 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 şi 0 aşa cum se procedează în sistemul zecimal (în baza zece). Amintiţi-vă că în cadrul sistemului de numeraţie în baza 10 fiecare poziţie dintr-un număr are asociată o anumită pondere. În cazul numărului 375, poziţia cifrei 5 este asociată unităţilor, cifra 7 se află pe poziţia zecilor, iar 3 pe poziţia sutelor

(figura 1.14a). Fiecare poziţie are asociată o pondere de zece ori mai mare decât poziţia din dreapta sa. Valoarea reprezentată de întreaga expresie se obţine înmulţindu-se valoarea fiecărei cifre cu ponderea asociată poziţiei ocupată de cifra respectivă în cadrul numărului şi apoi adunându-se rezultatul înmulţirtilor. De exemplu, $375 = 3 \text{ X } 10^2 + 7 \text{ X } 10^1 + 5 \text{ X } 1x10^0$.

În notaţia binară, poziţia fiecărei cifre este de asemenea asociată cu o anumită pondere, numai că ponderea asociată fiecare poziţii este de două ori mai mare decât ponderea poziţiei din dreapta. Mai exact, cifra cea mai din dreapta a unei reprezentări în binar are asociată ponderea unu (2^0), următoarea poziţie din stânga este asociată cu doi (2^1), următoarea are ponderea patru (2^2), următoarea este asociată cu opt (2^3), şi aşa mai departe. De exemplu, în cazul reprezentări 1011 în binar, cifra 1 cea mai din dreapta este asociată cu 1, următoarea cifră 1 se află în poziţia asociată cu doi, cifra 0 este asociată cu patru, iar cifra 1 cea mai din stânga este asociată lui opt (figura 1.14b).

Figura 1.15 Decodificarea reprezentării binare 100101



Pentru a afla valoarea corespunzătoare unei reprezentări în binar, vom proceda la fel ca şi în baza zece - vom înmulţi valoarea fiecărei cifre cu ponderea asociată poziţiei sale şi vom aduna rezultatele. De exemplu, valoarea reprezentată de 100101 este 37, după cum este arătat în figura 1.15. Observaţi că de vreme ce notaţia binară utilizează numai cifrele 0 şi 1, acest proces de înmulţire şi adunare se reduce doar la adunarea ponderilor asociate poziţiilor ocupate de valoarea 1. Astfel, modelul de biţi 1011 reprezintă valoarea unsprezece, fiindcă cifrele 1 se află în poziţiile asociate cu ponderile unu, doi şi opt.

Observaţi că secvenţa reprezentărilor binare obţinute prin numărarea de la zero la opt este următoarea:

0
1
10
11
100
101
110
111
1000

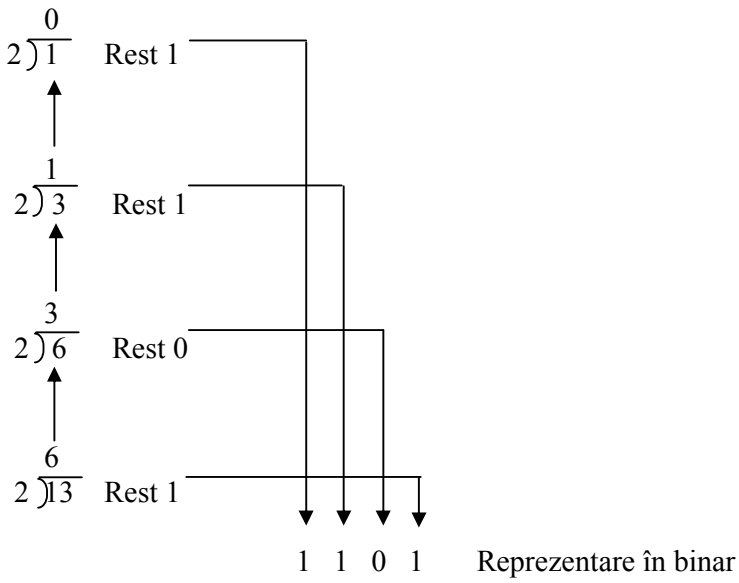
Există numeroase metode care pot fi utilizate pentru generarea acestei secvenţe şi, deşi nu sunt elegante în ceea ce priveşte partea teoretică, ele furnizează o metodă rapidă pentru obţinerea reprezentării în binar a valorilor mici. Una dintre metode este aceea de a ne imagina contorul de kilometraj al unei maşini ale cărui roţi de afişaj conţin numai cifrele 0 şi 1. Contorul porneşte din 0 şi se roteşte la 1 pe măsură ce maşina se deplasează. Apoi, în timp ce roata contorului trece din 1 în 0, ea va provoca apariţia unui 1 la stânga sa, producând afişarea lui 10. Valoarea 0 din dreapta se roteşte apoi în 1, producând 11. În acest moment, trecerea valorii 1 din dreapta în 0 va face ca valoarea 1 din stânga să treacă şi ea în zero. Acest fapt va avea ca efect apariţia altei cifre 1 în cea de a treia coloană şi deci afişarea valorii 100.

Figura 1.16 Algoritm pentru aflarea reprezentării în binar a unui număr întreg pozitiv

Pasul 1: Se împarte valoarea la doi şi se memorează restul împărţirii.
Pasul 2: Cât timp câtul obţinut este diferit de zero, se continuă împărţirea noului cât la doi memorându-se restul.
Pasul 3: Când s-a obţinut un cât egal cu zero, reprezentarea în binar a valorii iniţiale constă din resturile împărţirilor, afişate de la stânga în care au fost memorate.

Pentru a afla reprezentarea în binar a valorilor mari, veţi prefera probabil abordarea mai sistematică descrisă de algoritmul din figura 1.16. Să aplicăm acest algoritm asupra valorii treisprezece (figura 1.17). Vom împărţi mai întâi treisprezece la doi obţinând câtul şase şi restul unu. Deoarece câtul nu este egal cu zero, pasul 2 al algoritmului ne indică să îl împărţim la doi, obţinând un nou cât cu valoarea trei şi restul zero. Noul cât este de asemenea diferit de zero, aşa că îl vom împărţi din nou la doi, obţinând câtul 1 şi restul 1. Vom împărţi din nou şi acest cât la doi şi obţinem câtul zero şi restul unu. Cum am obţinut un cât cu valoarea zero, vom trece la pasul 3 al algoritmului, unde vom afla că reprezentarea în binar a valorii iniţiale (13) este 1101.

Figura 1.17 Aplicarea algoritmului din figura 1.12 pentru a se determina reprezentarea în binar a numărului treisprezece



Să revenim la problema inițială, stocarea datelor numerice. Utilizând notația în binar, putem memora într-un octet orice număr întreg între 0 şi 255 (între 00000000 şi 11111111), iar dacă utilizăm doi octeți putem stoca numere întregi aflate 0 şi 65535. Acest fapt este o îmbunătățire spectaculoasă față de posibilitattea de a memora doar numere întregi de la 0 la 99, ca în cazul codificării caracterelor ASCII.

Din această cauză, precum şi din alte motive, se obişnuieşte ca informațiile numerice să fie stocate utilizând o formă binară în locul codificării simbolurilor. Spunem ″o formă a notației binare″, deoarece sistemul binar simplu pe care l-am descris reprezintă doar fundamentul mai multor tehnici de stocare utilizate în domeniul calculatoarelor. Unele variante ale sistemului binar sunt discutate mai târziu în acest capitol. Deocamdată vom reține că pentru stocarea numerelor întregi se utilizează de obicei un sistem denumit notația notația în complement față de doi, deoarece acesta pune la dispoziție o metodă comodă pentru reprezentarea atât a numerelor pozitive cât şi a celor negative. Pentru reprezentarea numerelor care conțin părți fracționare ca de exemplu 4 1/2 sau 2 3/4, se utilizează altă tehnică, ce poartă numele de reprezentare în virgulă mobillă. În concluzie, o anumită valoare (cum ar fi 25) poate fi reprezentată prin diferite modele de biți (sub forma unor coduri de caractere, în notația în complement față de doi sau în notația în virgulă mobilă ca 25 0/2); reciproc, un anumit şir de biți poate fi interpretat în diferite moduri.

Să menționăm acum altă problemă importantă care apare la sistemele de stocare a valorilor numerice. Indiferent de lungimea şirului de biți pe care-l poate aloca un calculator pentru stocarea valorii numerice, vor exista totuşi valori prea mari sau părți fracționare prea mici pentru a putea fi memorate în spațiul alocat. Ca rezultat va exista întotdeauna riscul să apară erori cum ar fi depăşirea superioară (valori prea mari) sau eroarea de rotunjire (fracții prea mici), erori care trebuie tratate, pentru a nu se ajunge la situația ca un utilizator al calculatorului să fie pus, fără a bănui nimic, în fața unor date eronate.

1.3.3 Reprezentarea altor tipuri de date

Aplicațiile utilizate la calculatoarele de astăzi implică şi folosirea altor tipuri de date decât caractere sau numere. Ele utilizează imagini, sunete, secvențe video. Comparativ cu sistemele de stocare a caracterelor şi valorilor numerice, tehnicile utilizate pentru reprezentarea acestor tipuri de date suplimentare se află abia la început şi deci nu există încă o standardizare global acceptată de întreaga comunitate informatică.

Una dintre metodele răspândite de stocare a imaginilor este considerarea imaginii ca o colecție de puncte, numite **pixel**, prescurtarea sintagmei “picture element“ (element de imagine). În forma cea mai simplă, o imagine alb-negru poate fi codificată ca un lung şir de biți ce reprezintă liniile de pixeli din imagine, unde fiecare bit are valoarea 1 sau 0, în funcție de culoarea pe care o are pixelul asociat lui (negru sau alb). Reprezentarea imaginilor color este doar puțin mai complicată, deoarece fiecare pixel poate fi reprezentat de o combinație de biți care să îi indice culoarea.

Reprezentările de acest tip sunt cunoscute sub numele de **hărți prin biți** (**bit maps**),ceea ce înseamnă că şirul de biți nu este mic mai mult decât o hartă a imagini reprezentate. Sistemele populare de hărți de biți include TIFF (Tag Image Format File) şi GIF (Graphic Interchange Format). Fotografiile sunt adesea reprezentate într-o hartă de biți cunoscută ca JPEG (Joint Photographic Experts Group). Dezavantajul principal al acestor sisteme este acela că imaginea nu poate fi scanată la o dimensiune arbitrară. Ea poate fi tipărită la imprimantă

utilizând un pixel de imprimantă pentru a reprezenta un pixel original sau eventual blocuri de 2x2 pixeli de imprimantă pentru a reprezenta un pixel din imagine. Ultima metodă va produce o imagine mai mare decât prima, dar o scalare intermediară ar fi dificil de realizat.

Pentru a rezolva problema scalării, imaginea poate fi memorată ca un set de directive care precizează modul de desenare al imaginii, în loc să se utilizeze o reprezentare pixel cu pixel. De exemplu, o linie existentă în imagine ar putea fi reprezentată prin instrucțiunea de desenare a unei linii între două puncte anumite. O asemenea metodă lasă detaliile referitoare la modul de desenare a liniei în seama dispozitivului care produce imaginea în loc să-i impună acestuia să reproducă un model particular de pixeli. Ea furnizează de asemenea o descriere compatibilă cu orice mărime a unităților sistemului de coordonate care ar putea fi specificat atunci când se doreşte afişarea imaginii.

Diferitele fonturi disponibile pentru imprimantele şi monitoarele de astăzi sunt reprezentate de obicei în acest mod, ceea ce oferă flexibilitate la redimensionarea caracterelor; ele se numesc **fonturi scalabile** (**scalable font**). De exemplu, True Type ® (dezvoltat de Microsoft şi Aplle Computer) este un sistem care indică modul în care să fie desenate simbolurile din text. De asemenea, PostScript ® (dezvoltat de Adobe Systems) pune la dispoziție o metodă de descriere a caracterelor şi în general a datelor grafice.

De asemenea, tot metode de reprezentare a datelor sunt şi MPEG (Motion Picture Experts Group), o tehnică folosită pentru datele video şi audio, şi DXF (Drawing Interchange Format), utilizat la sisteme de proiectare asistată de calculator (Computer Aided Design – CAD) în care imaginile trebuie rotite şi redimensionate pe ecranul calculatorului.

1.4 Sistemul binar de numerație

Înainte să studiem tehnicile de stocare a valorilor numerice utilizate în calculatoarele de astăzi, trebuie să aflăm mai multe detalii despre sistemul de reprezentare binar.

Figura 1.18 Adunarea a două numere în binar

0	1	0	1
+0	+0	+1	+1
0	1	1	0

1.4.1 Adunarea în binar

Pentru a aduna două valori reprezentate în notație binară, vom începe, la fel cum am procedat cu baza zece în şcoala primară, prin memorarea regulilor de adunare (figura 1.18), care de altfel sunt utilizate la adunarea a două şiruri de cifre zecimale. Procedați astfel:

```
 00111010
+00011011
```

vom începe prin a aduna cifrele 0 şi 1 cele mai din dreapta; obţinem rezultatul 1, care-l vom scrie sub coloana respectivă. Vom aduna apoi cifrele 1 şi 1 din coloana următoare, obţinând rezultatul 10. Vom scrie 0 sub coloană şi vom transfera cifra 1 deasupra coloanei următoare. În acest moment, situaţia arată astfel:

```
      1
  00111010
+ 00011011
  --------
        01
```

Vom aduna cifrele 1, 0 şi 0 din coloana următoare, obţinând rezultatul 1 şi deci vom scrie 1 sub coloană. Cifrele 1 şi 1 din următoarea coloană dau ca rezultat al adunării 10; vom scrie 0 sub coloana respectivă şi vom transfera cifra 1 deasupra coloanei următoare. În acest moment, situaţia arată astfel :

```
     1
  00111010
+ 00011011
  --------
      0101
```

Adunarea cifrelor 1,1 şi 1 din coloana următoare dă ca rezultat 11; vom scrie cifra 1 sub coloana respectivă şi vom transfera cifra 1 deasupra coloanei următoare. Vom aduna acest 1 cu cifrele1 şi 0 existente deja în această coloană şi vom obţine 10. Din nou vom scrie cifra 0 sub coloană şi vom transfera 1 în coloana următoare. Avem acum:

```
    1
  00111010
+ 00011011
  --------
    010101
```

Vom aduna acum cifrele 1, 0 şi 0 din penultima coloană, obţinând rezultatul 1, pe care-l vom scrie sub coloană, fără a avea ce transfera către coloana următoare. În sfârşit vom aduna cifrele din ultima coloană, obţinând rezultatul 0 şi-l vom scrie sub această coloană. Rezultatul final al adunări este următorul:

```
  00111010
+ 00011011
  --------
  01010101
```

ANEXA A

Codul ASCII

În continuare se prezintă o parte dintre codurile ASCII; fiecare cod binar a fost extins prin adăugarea în stânga a unui bit cu valoarea 0, pentru a se obține codurile pe opt biți utilizate în prezent.

Simbol	*Cod ASCII*	*Simbol*	*Cod ASCII*	*Simbol*	*Cod ASCII*
(space)	00100000	?	00111111	^	01011110
!	00100001	@	01000000	_	01011111
"	00100010	A	01000001	a	01100001
#	00100011	B	01000010	b	01100010
$	00100100	C	01000011	c	01100011
%	00100101	D	01000100	d	01100100
&	00100110	E	01000101	e	01100101
'	00100111	F	01000110	f	01100110
(	00101000	G	01000111	g	01100111
)	00101001	H	01001000	h	01101000
*	00101010	I	01001001	i	01101001
+	00101011	J	01001010	j	01101010
,	00101100	K	01001011	k	01101011
-	00101101	L	01001100	l	01101100
.	00101110	M	01001101	m	01101101
/	00101111	N	01001110	n	01101110
0	00110000	O	01001111	o	01101111
1	00110001	P	01010000	p	01110000
2	00110010	Q	01010001	q	01110001
3	00110011	R	01010010	r	01110010
4	00110100	S	01010011	s	01110011
5	00110101	T	01010100	t	01110100
6	00110110	U	01010101	u	01110101
7	00110111	V	01010110	v	01110110
8	00111000	W	01010111	w	01110111
9	00111001	X	01011000	x	01111000
:	00111010	Y	01011001	y	01111001
;	00111011	Z	01011010	z	01111010
<	00111100	[	01011011	{	01111011
=	00111101	\	01011100	}	01111101
>	00111110	]	01011101		