CAPITOLUL 1

1. Introducere

Calculatoarele electronice au apărut în jurul anului 1940. Totuşi unele maşini mecanice de calculat au fost propuse anterior:

- Pascal 1641 – realiza numai adunare şi scădere
- Leibnitz 1674 – realiza şi înmulțire
- Babbage 1840 – maşină analitică

Chiar şi în aceste aparate primitive unele din componentele unui arhitecturi moderne pot fi găsite. De exemplu un registru temporar de memorie care menține variabilele aflate în curs de procesare. Acest registru este cunoscut sub denumirea de registru acumulator. Calculatoarele electronice au fost inițial propuse sub forma unui aparat abstract care poate realiza sau simula orice tip de maşină mecanică. Forma teoretică a fost introdusă de Alan Turing pe la mijlocul anului 1930.

Ca şi în cazul multor alte tehnologii, dezvoltarea practică a unei maşini care să respecte acest model, a fost inflențată de al doilea război mondial. Acest lucru a fost posibil deoarece se dorea calcularea traiectoriilor de rachete (Collosus şi Eniac) sau decodarea codului Enigma folosit de Germania pentru protecția mesajelor.

Primul program stocat electronic a fost folosit în 1948. În primii ani de dezvoltare (1950) computerele şi impactul lor ulterior în viața umană au fost mult subevaluate. E de notorietate declarația lui Bill Gates, fondatorul Microsoft cum că nu vede cum ar putea fi vreo dată nevoie de mai mult de 640 Ko RAM.

În anii ’70 a apărut pastila de siliciu şi tehnologia VLSI (Very large Scale Integration) ca tehnică de producție fapt ce adus la computere ieftine şi rapide. Acest lucru a condus la creşterea şi diversificarea rapidă a categoriilor de utilizatori care ăşi puteau permite să le achiziționeze.

Perioada 1970-1980 care este caracterizată de prin apariția microsistemelor pe 8 biți, care deşi erau scumpe permiteau accesul separat de *main-frame* la posibilitățile sistemului de calcul. În această perioadă este demnă de remarcat apariția în 1973 a lui I8080 care poate fi considerată un punct de referință în domeniu. Din acest moment softul a început să fie dezvoltat nu numai pentru calcule extrem de complexe ci şi pentru jocuri, editoare de texte, programe de baze de date şi alte facilități cerute de piață.

Tot în această perioadă, forțată şi de ruperea de main-frame, a pornit şi dezvoltarea sistemelor de operare "locale" CPM şi apoi DOS.

Sistemul de operare MS-DOS a fost realizat prin adaptarea la noile instrucțiuni maşină a unui *microkernel* de UNIX la care s-au adăugat funcțiile necesare controlului perifericelor care începuseră să fie ataşate calculatoarelor personale.

Din 1980 până în 1985, deşi încă foarte scumpe, au început să apară pe piață arhitecturi din ce în ce mai puternice pe 16, 32 biți iar aria lor de folosire s-a extins foarte mult şi în alte domenii.

Nu este de neglijat nici impactul produs de interfețele vizuale şi de controalele prin zone active ale ecranului. Aceste controale care sunt activate din *mouse* sunt caracteristice sistemului de operare WINDOWS.

Tot în această perioadă, din raţiuni economice şi practice, s-a produs dispariţia microcalculatoarelor acestea fiind mult depăşite de cerinţele pieţii.

Din 1985 se poate spune că s-a intrat într-o nouă eră în acest domeniu prin ieftinirea dramatică a hardware-ului şi apariţia reţelelor de calculatoare.

O dată cu ieftinirea arhitecturilor de calcul s-a putut divesrifica şi mai mult aria de utilizare a lor. Acest lucru a avut ca rezultat o creştere a preţului softului.

Conceptul de reţea permitea transferuri rapide de date , a căror cantitate a început să fie din ce în ce mai mare.

Iniţial majoritatea reţelelor de calculatoare erau "închise " în sensul că nu comunicau între ele. Apoi în ultima perioadă termenul de sistem "deschis" capătă din punct de vedere practic o răspândire din ce în ce mai mare la nivel global. Din acest punct de vedere reţele de tip Arpanet , Internet ş.a.m.d sunt reprezentative.

În ultimii ani însuşi conceptul de transmisie de informaţie interumană tinde să se schimbe. Televiziunea digitală este deja implementată , iar preţul perifericelor dedicate a scăzut extraordinar astfel încât se preconizează dezvoltarea calculatorului astfel încât el să nu mai fie strict orientat spre aplicaţii dedicate ci să fie o legătură dinamică de orice tip (audio, video, teleworking) a individului cu societatea.

Vom prezenta câteva din aplicaţiile mai cunoscute ale calculatoarelor în diverse domenii:

- Industrie: control şi automatizare → microcontrolere, CAM (Computer Aided Manufactory);
- Economie: gestiune, transferul banilor, "banii electronici" .
- Medicină: diagnoză automată, cercetarea electronică la diverse nivele ale organismului uman, analiza codului genetic, diagnoză automată → reţele neurale, staţii grafice, sisteme expert.
- Pictură: noi ramuri ale picturii folosind computerul, restaurare → holografie reţele neurale, procesoare analogice.
- Muzică şi televiziune: prelucrări de imagini şi sunete practic de orice natură → procesoare analogice, staţii grafice, arhitecturi MMX .
- Proiectare : programele de tip CAD (Computer Aided Design)

Se mai pot da exemple în multe domenii, însă este cert că la ora actuală viaţa şi societatea umană nu mai pot fi concepute fără calculator.

1.1. Programarea şi limbaje de programare

Prin **programare** se înțelege în mod generic transpunerea unor operații repetitive, asupra unui set de date, într-un limbaj inteligibil de către un sistem de calcul care urmează ulterior să le execute. Acest lucru este realizat în două etape

- etapă în care este implicat omul şi anume cea de trecere de la problema reală la transpunerea într-un limbaj de programare.
- o a doua etapă, automată, care transpune **codul sursă (**înşiruirea de instrucțiuni specifice limbajului respectiv**)** într-un **cod direct executabilă (**inteligibil sistemului de calcul) lucru de care se ocupă programe specializate numite **compilatoare**.

Rolul programării este ca fie dată o anumită operațiune sau suită de operațiuni repetitivă(e) care se aplică asupra unor seturi de date mereu diferite să fie scris un program care să ceară setul de date **de intrare**(cele care trebuie să fie prelucrate) să execute asupra lor suita standard de operațiuni şi să livreze **datele de ieşire (**adică rezultatele).

În aceste condiții programul trebuie să fie compus din mai multe instrucțiuni. În primul rând cele care **rezervă** (declară o zonă de memorie ca fiind special dedicată numai pentru anumite date), o zonă de memorie în care se vor citi datele de intrare, o zonă de memorie în care se vor păstra datele **intermediare** (cele care apar în decursul calculelor dar de care utilizatorul nu are nevoie) şi de asemeni pe cele de ieşire. Programul trebuie să conțină instrucțiuni care să citească de la utilizator datele de intrare, urmate în sfârşit de instrucțiunile de calcul propriu zis ca în final să conțină instrucțiunile de afişare a rezultatelor (datele de ieşire)

1.2 Arhitectura unui sistem de calcul

Ca ***observație*** de bază: un sistem de calcul este pur şi simplu versiunea mult dezvoltată a unui calculator de buzunar şi deci nu trebuie să ne aşteptăm de la el ca vreo dată să facă altceva decât l-a pus cineva, un programator prin intermediul unui program, să facă. "Inteligența" aparentă a unui sistem de calcul provine din cantitatea de inteligență umană investită în respectivul program aflat în execuție.

Hilar, am putea spune că omul încearcă să "facă pe cineva după chipul şi asemănarea lui" dar rezultatul este o copie palidă care altfel este extrem de utilă. Din aceste lucruri derivă vestitul principiu de aur al primilor programatori GIGO (Garbage In Garbage Out), care într-o traducere prozaică înseamnă: "gunoi bagi gunoi scoți" şi care are meritul, în afară de cel umoristic, de a atrage atenția oricui care se va apuca de programare că din nefericire calculatorul face numai ce i se indică prin diverse metode să facă. De aici rezultă o primă atenționare valabilă aprope tot timpul în cazul **depanării**(execuției pas cu pas în scopul găsirii eventualelor defecte în funcționare) unui program : CAUTĂ ÎNTÂI EROAREA TA DE PROGRAMATOR ŞI APOI DĂ VINA PE CALCULATOR.

Desigur există un procent cam de 5% din situații în care un program scris corect este executat greşit, lucru care apare ca rezultat al interferenței unor programe neautorizate, cum ar fi viruşii, cu programul aflat în execuție. În cazul în care nu este

valabil acest lucru problema este mai complexă putând fi implicate defecțiuni aleatorii ale părții hardware. Oricum ar fi aceste lucruri intră în sfera de acțiune a unui inginer de sistem, şi deci nu sunt luate aprioric în considerație fiind de obicei rare.

Pentru a putea înțelege arhitectura vom căuta anumite analogii între un om şi un sistem de calcul.

Să analizăm ce se întâmplă cu un om în momentul în care vede o formulare de genul : 2*2=… .

Deşi ni se pare ceva evident, cu toții am învățat pe de rost tabla înmulțirii să încercăm acum o descompunere a paşilor care sunt executați.

În primul rând memorăm în memoria temporară (sau de scurtă durată) **operanzii(**datele asupra cărora vom executa o operațiune**)** apoi realizăm operațiunea de înmulțire rezultând 4, care este până la comunicarea lui verbală sau prin scris tot în memoria temporară. Se observă că operanzii 2,2 şi operațiunea * au fost preluate de pe un suport extern si anume cel scris, rezultatul putând fi **returnat(**trimis către emițătorul unei cereri de lucru **),** după cum am spus tot pe acelşi sistem extern de stocare.

Acest exemplu nu a fost dat fără un scop şi anume, rolul lui este printre altele de a vă obişnui cu nivelul extrem de scăzut la care se face analiza şi descompunerea unei probleme reale în momentul în care doresc să o transpun într-un limbaj de programare.

Şi acum să analizăm **arhitectura bloc** (structura funcțională) a unui sistem de calcul.

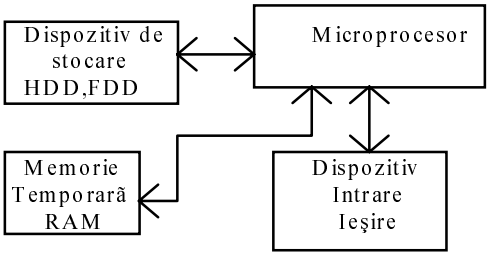

Fig. 1 Arhitectura unui sistem de calcul

După cum se observă din fig. 1 într-un sistem de calcul există:

- un dispozitiv de stocare în masă, rolul caietului pe care scrim /citim în cazul nostru. Acest dispozitiv este materializat prin Hard Disk (HDD) căruia i se mai spune şi suport fix de date şi Floppy Disk (FDD) căruia i se mai spune şi suport mobil de date .
- o memorie temorară, RAM (Random Acces Memory) care joacă exact rolul memoriei de scurtă durată umane.
- dispozitive de Intrare sau de Ieşire prin care sistemul primeşte sau returnează date. În cazul unui sistem de calcul acestea sunt tastatura şi mouse ca dispozitive de intrare şi monitorul sau imprimanta ca dispozitive de ieşire.

A mai rămas de lămurit problema microprocesorului care este "inima" unui sistem de calcul şi are rolul de a coordona întregul sistem. Desigur că această coordonare se realizează tot prin intermediul unor programe care există în ceea cea am putea numi, prin analogie cu omul, partea de reflexe condiționate, aceste programe se găsesc într-o memorie ROM (Read Only Memory) deci nu pot fi decât citite.

1.3 Funcționarea unui sistem de calcul

Sistemul de calcul după cum se observă comportă destul similitudini cu omul la nivel de arhitectură şi deasemeni, după cum vom discuta, şi la nivel de funcționare.

Să vedem cum se realizează, bineînțeles la nivel simplificat, execuția unei formule de genul 2*2=… .

Se citeşte de pe suportul de stocare programul care conține cererea noastră, se încarcă în memoria temporară (de acest lucru se ocupă **sistemul de operare(**programul specializat care se ocupă de traducerea unor instrucțiuni simple şi intuitive ale utilizatorului în comenzi directe către partea de execuție a sistemului**)**, în urma cererii noastre explicite de lansare a programului) şi se începe execuția lui de către procesor execuție care urmează aproximativ sistemul uman de gândire. Pe scurt modul de execuție la nivelul procesorului poate fi descris ca :

- citeşte datele utile (din RAM în aceast caz) (2,2)
- citeşte operația care trebuie executată asupra datelor suspomenite(tot din RAM)(*)
- execută operația (2*2=4)
- returnează rezultatul (înnapoi în RAM)(4).

La nivelul sistemului de calcul modul de execuție arată ca mai jos:

- citeşte programul de executat (de pe sistemul de stocare fix/mobil) şi
- încarcă-l în RAM
- execută programul (din RAM)
- la terminarea programului aşteaptă noua cerere de execuție

Memoria Temporară a unui sistem de calcul

După cum am pomenit în primul capitol, una din componentele de bază ale unui sistem de calcul este memoria dinamică RAM sau memoria temporară.

Această memorie a fost concepută ținând cont de faptul că sistemul de numerație folosit în interiorul unui calculator este bază 2. De aici rezultă că orice fel de dată sau instrucțiune efectiv procesată sau executată la nivelul maşinii trebuie să fie pastrată sub forma ei binară (deci corespunzătoare bazei 2). Întrucât în baza 2 nu există decât două numere 0 şi 1 rezultă că vom avea reprezentări sub forma unor şiruri de 0 şi de 1. La nivelul maşinii materializarea efectivă a respectivelor numere este realizată, de obicei, prin ataşarea logică a lui "1" la o tensiune de 5 volți iar lui "0" lipsei acestei tensiuni. Totuşi indiferent de nivelul tehnologic este absolut nepractic ca să considerăm fix 5 V sau fix 0 V ca valori de comparație pentru că acest lucru ar fi dus la nişte echipamente extrem de scumpe şi de lente. Din aceasta cauză se consideră a fi 0 sau 1 logic în jurul valorilor sus menționate cu $\pm\Delta V$. De aici a rezultat posibilitatea folosirii unui condensator ca element de memorare. Acest lucru este posibil deoarece în cazul unei reîncărcări periodice a acestuia se poate obține ca tensiunea la bornele lui să varieze în limitele necesare nouă dacă fac suficient de repede această reîncărcare. În aceste condiții foarte simplist putem vedea o memorie dinamică ca o matrice imensă de condensatoare care pot fi încărcate sau descărcate în cursul operațiunilor de scriere/citire.

Aici mai apare o problemă şi anume numărul de biți care compun datele utile de calcul (**bit** unitate informațională corespunzătoare existenței sau a unei tensiuni la intrarea unui dispozitiv digital) care pot fi procesați la un moment dat de un procesor.

Întrucât memoria RAM este cea care furnizează respectivele date era logic ca ea să fie organizată în diviziuni de o mărime egală cu cea a numărului de biți sus menționați. Aceste grupări se numesc **locații de memorie** şi pot fi considerate ca minima unitate care poate fi scrisă sau citită dintr-o memorie. O dată lămurit acest aspect devine clar că pentru a mă putea referi la una sau alta din locații trebuie să existe o metodă de individualizare strictă. S-a ales cea mai simplă cu putință şi anume
numerotarea lor numărul locației numindu-se **adresa locației**.

1.4. Noțiunea de algoritm

În procesul de rezolvare a unei probleme folosind calculatorul există două faze:

- definirea şi analiza problemei;
- proiectarea şi implementarea unui algoritm care rezolvă problema.

Definirea şi analiza problemei poate fi la rândul ei descompusă în:
- specificarea datelor de intrare;
- specificarea datelor de ieşire.

Specificarea datelor de intrare constă în:

1. Ce date vor fi primite la intrare;
2. Care este formatul (forma lor de reprezentare) a datelor de intrare;
3. Care sunt valorile permise sau nepermise pentru datele de intrare;
4. Există unele restricții (altele decât la 3) privind valorile de intrare;
5. Câte valori vor fi la intrare, sau dacă nu se poate specifica un număr fix de valori, cum se va şti când s-au terminat de introdus datele de intrare.

Specificarea datelor de ieşire trebuie să țină cont de următoarele aspecte:

1. Care din valorile rezultate în cursul aplicării algoritmului de calcul, asupra datelor de intrare, vor fi afişate (necesare utilizatorului). În acest pas se face diferențierea clară între date intermediare şi date de ieşire.
2. Care va fi formatul datelor de ieşire (de exemplu un număr real poate fi afişat cu trei sau cu cinci zecimale, sau un text poate fi afişat integral sau parțial)
3. Sunt sau nu necesare explicații suplimentare pentru utilizator în afara datelor de ieşire.
4. Care este numărul de date de ieşire care trebuie transmise către ieşire.

O definiție a noțiunii de algoritm poate fi: înlănțuirea de paşi simpli, operații distincte care descriu modul de prelucrare a unor date de intrare în scopul rezolvării unei probleme. Deci, ALGORTIMUL ESTE ACEA PARTE DE CALCUL COMUNĂ ÎN CAZUL REZOLVĂRII UNEI PROBLEME DE CALCUL.

Un exemplu simplu de algoritm ar fi suita de operații matematice făcută în rezolvarea unei ecuații matematice de gradul II $aX^2+bX+c=0$, coeficienții a, b, c se schimbă dar modul de procesare a valorilor lor, nu.

Proprietățile unui algoritm sunt:

1. Este compus din instrucțiuni simple şi clare.
2. Operațiunile specificate de instrucțiuni se execută într-o anumită secvență.
3. Soluția trebuie obținută într-un număr finit de paşi.

Din cele spuse mai sus rezultă că UN ALGORITM ESTE INDEPENDENT DE TIPUL DE LIMBAJ ÎN CARE ESTE TRANSPUS SAU DE TIPUL DE MAŞINĂ PE CARE ESTE EXECUTAT.

Observație: În această carte se vor discuta numai ALGORITMI SECVENȚIALI şi respectiv PROGRAMAREA SECVENȚIALĂ.

Un algoritm secvențial este acel algoritm creat pentru execuția pe un calculator secvențial. Denumirea de secvențial provine din faptul că acesta nu poate executa decât o singură operație la un moment dat.

1.5. Reprezentarea algoritmilor

În general, un algoritm poate fi considerat ca o descriere a prelucrării efectuate asupra unui flux de date, prelucrare care are loc cu un scop bine determinat. Ca de obicei, în cazul dezvoltării unei noi teorii, s-a căutat atât o cale de standardizare a modului de descriere a unui algoritm, cât şi o cale de a-l face independent de un limbaj de programare anume rezultând două metode clasice:

- metoda schemei logice (diagramei de flux (flowchart));
- metoda pseudocodului.

1.5.1. Metoda schemei logice

În cadrul acestei metode se foloseşte un set de simboluri, prezentat în figura 1.1, pentru descrierea paşilor ce trebuie executați pentru ca programul rezultat să ne rezolve o anumită problemă.

Deşi a fost extrem de folosită, până nu de mult, această metodă pare a pierde teren în fața reprezentării de tip pseudocod, poate şi datorită timpului suplimentar pierdut de utilizator cu executarea simbolurilor grafice.

Să analizăm un program de calcul a mediei pentru trei note şi să vedem cum ar apărea descris prin această metodă.

După cum se observă în figura 1.2 se descriu efectiv pas cu pas toate operațiunile care trebuiesc urmate.

În figura 1.2 s-a specificat şi începutul programului, fapt care pare inutil, dar în cazul descrierii unui program foarte mare se prea poate ca schema logică să se întindă pe zeci de pagini, caz în care devine evidentă necesitatea declarării începutului şi sfârşitului de program.

În următorul pas se declară variabilele. O variabilă este un identificator propriu al programatorului, care în cazul operațiunii de declarare se ataşează, la generarea programului, unei zone de memorie a cărei dimensiune este în directă concordanță cu tipul de dată stocat de aceasta.

Se trece apoi la citirea datelor de intrare. În acest caz este vorba de trei note ceea ce presupune că valorile de intrare pot avea şi cu zecimale. Mai târziu se va discuta despre alegerea tipului de variabilă funcție de specificațiile utilizatorului.

Se execută calculul propriu zis, după care se afişează rezultatul cu 2 zecimale. A mai rămas doar marcarea sfârşitul de program. Chiar dacă metoda are o impresie "artistică" sporită câte o dată se pierde prea mult timp şi hârtie în cazul folosirii ei şi de aceea din ce în ce mai mulți preferă ca metodă de descriere pseudocodul.

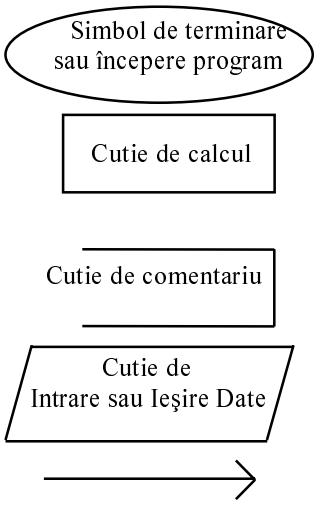

Fig. 1.1. Simboluri standard pentru metoda schemei logice

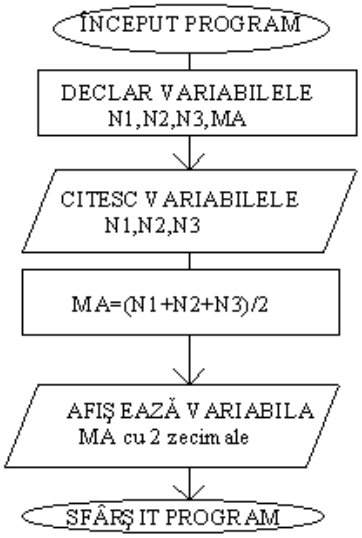

Fig. 1.2. Schema logică unui program de calcul a mediei pentru trei note

1.5.2. Metoda pseudocodului

Față de schema logică are dezavantajul că nu întotdeauna redă absolut toți pașii de calcul. Totuşi este mai comodă de folosit în cazul în care se descriu algoritmi pentru limbajele de programare de nivel mediu şi înalt.

Schema logică este cea mai indicată ca metodă de descriere, în special, în cazul implementării unui algoritm folosind limbaj de asamblare, unde descrierea amănunțită a fiecărui pas este necesară pentru putea minimiza şansele apariției unor erori de implementare.

Descrierea efectivă a paşilor ce trebuie efectuați, folosind cuvintele limbii române împreună cu cuvinte cheie ale limbajului de programare în care se va implementa algoritmul se numeşte pseudocod. Se observă că în acest caz criteriul independenței totale a descrierii față de limbajul de implementare nu este întru totul respectat.

Pentru comparație vom descrie mai jos în pseudocod problema prezentată în figura 1.2.

```
variabile: ma,n1,n2,n3
{
citeşte n1,n2,n3
ma=(n1+n2+n3)/3
afişează ma cu 2 zecimale
}
```