Universitatea „Ovidius” Constanța
Facultatea de Matematică şi Informatică

ARHITECTURA CALCULATOARELOR
(suport de curs şi laborator)

lect. Ozten CHELAI

Obiective:

Datorită accesibilității calculatoarelor pe de o parte, din punct de vedere al utilizatorului şi al instrumentelor pe care acestea le oferă şi dinamicii care există în domeniul informaticii, pe de altă parte, obiectivele cursului "Arhitectura sistemelor de calcul" sunt:

- asimilarea de cunoştințe fundamentale, la nivel conceptual, privind arhitectura sistemelor de calcul
- dobândirea de cunoştințe generale prinvind serviciile oferite de sistemele de calcul
- cunoşterea sistemului la nivel hardware şi programarea la nivelul maşinii (limbaj de asamblare)

Mențiuni

Prezentul suport cuprinde o bază de studiu pentru studenți, dar nu garantează o bună înțelegere a cunoştințelor prezentate. Materialul întocmit nu cuprinde o descriere în detaliu şi cu accentele necesare aprofundării. Aceasta este o primă formă de prezentare a cursului, cu erorile şi scuzele de rigoare, care va fi ulterior îmbunătățită, restructurată şi adăugită. Pentru o înțelegere temeinică a cunoştințelor prezentate, sfătuiesc studenții să vină la curs.

CUPRINS

1. DESCRIERE GENERALĂ A SISTEMULUI DE CALCUL

1.1. CONCEPTE GENERALE

1.1.1. Noțiunea de sistem

Sistem (definiţie) = Un ansamblu de elemente inter-relaționate ce compun un întreg.
Termenul de „system" in latină şi greacă înseamnă „a pune împreună, a combina".
Un **subsistem** este o parte a unui sistem.
În mod tipic un sistem este alcătuit din componente (elemente) care sunt interconectate şi interacționează pentru a facilita fluxul de informaţie.
În funcţie de tipul sistemului acesta se poate diferenția de elemente, maşini, procese.

Tipuri de sisteme:

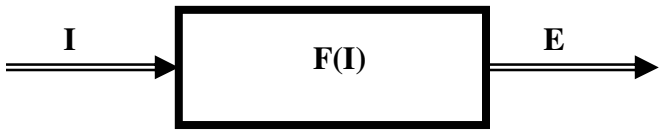
- **Sisteme deschise** – Sisteme care pot fi influențate de evenimentele din afara granițelor lor.
- **Sisteme închise** - Sisteme care nu sunt influențate de evenimentele din afara granițelor lor.
- **Sisteme dinamice** – Sisteme cu componente sau fluxuri care se **schimbă în timp.**

O distincție care trebuie specificată este între sistemele **fizice** şi cele **conceptuale**. Cele conceptuale sunt abstracte şi au la bază idei.ajutând la modelarea sistemelor fizice.

Arhitectura sistem = dispunerea si interconectarea componentelor pentru a obtine functionalitatea dorita a sistemului.

1.1.2. Conceptul de cutie neagră (black box)

Definitie. Un sistem cu intrari (I), iesiri(E) si transformari (F(x)) cunoscute, dar cu conținut necunoscut se numeste **black-box**.
Proprietatea cea mai importanta a cutiei negre este *utilizabilitatea*. I.e. utilizare fără a cunoaste detalii de implementare.



Conceptul de cutie neagra este utilizat in proiectarea si realizarea sistemelor de calcul. Sistemele sunt proiectate modular folosind cutii negre dupa urmatoarea regula: "ori de cite ori este necesara o functie intr-un sistem se foloseste o cutie neagra care realizeaza aceasta functie". Modul in care este implementata respectiva functie nu intereseaza pe utilizator, ci doar functionalitatea cutiei negre si modaliatatea de folosire a ei.

Pentru a facilita construcția sistemelor din module cu funcționalitate cunoscută (black box) acestea au fost **standardizate**.
Un **standard** cuprinde o descriere a modului de utilizare a unui modul (specificaţii de utilizare).

Organizațiile internationale de standarde: **ISO** (International Standard Organization), **IEEE** (Institute of Electrical and Electronics Engineers), **IETF** (Internet Engineering Task Force) au elaborat o serie de standarde, respectate de producatori, în realizarea modulelor respective.

1.2. SISTEMUL DE CALCUL.

1.2.1. Definiție

Un **sistem de calcul** este un sistem care execută programe stocate în memorie în interacțiune cu mediul extern.

Componentele sistemului de calcul sunt

- hardware – echipamente
- software - programe

1.2.2. Evoluția sistemelor de calcul

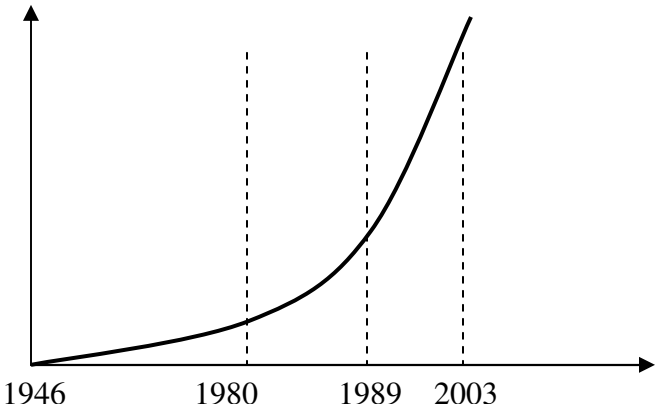
În anul 1946 John von Neumann introduce

- conceptul programarii calculatoarelor prin stocarea in memorie a datelor si programelor. Acest concept stă la baza calculatoarelor moderne
- utilizarea sistemului binar în locul celui zecimal

Calculatorul este de fapt o cutie neagră (black box) care înglobează o funcționalitate expusă utilizatorului sau sistemului extern. Expunerea funcționalității către utilizator se face într-un mod foarte sugestiv, grafic printr-o interfață de utilizare GUI (graphic User Interface).

Evoluție: se observă o crestere accentuata in ultimele doua decenii

- dupa1989 **exploziva**
- apar
  - **concepte noi**
  - **domenii noi**



Din punct de vedere al utilizatorului obişnuit progresul a fost realizat datorită:

- accesabilității
  - disponibilitatea calculatoarelor prin prețul mic al acestora
  - uşurința în utilizare – interfața grafică prietenoasă, sugestivă
- funcționalității crescute
  - prin creşterea resurselor de calcul şi memorare (viteză de execuție şi memorie)
  - înglobarea a mai multor funcții complexe
- suportului de comunicare oferit
  - prin folosirea infrastructurii existente de comunicație
  - posibilitatea de comunica diferite tipuri de informații: poştă, multimedia, etc.

1.2.3. Arhitectura sistemelor de calcul

Arhitectura sistemelor de calcul sau arhitectura calculatoarelor este **teoria** din spatele construcției unui calculator. În acelaşi mod în care un arhitect stabileşte principiile şi obiectivele construirii unui proiect ca baze ale unor planuri de construcție, în acelaşi mod un arhitect de calculatoare stabileşte arhitectura sistemului de calcul ca bază a specificațiilor de proiectare.

Obiectivul principal în arhitectura unui sistem de calcul îl reprezintă un **raport cost/performanță** cât mai bun.

Componenta sistem = cutie neagra.
Arhitectura sistem = dispunerea si interconectarea componentelor pentru a obtine functionalitatea dorita a sistemului.

Arhitecturi generale

1. **Arhitectura multistrat** – niveluri ierarhice. Un nivel inferior ofera suport nivelului superior pentru executia functiilor sale
2. **Decompozitia functionala** – descompunere a componentelor dupa functiile realizate
3. **Decompozitia conceptuala** – descompunere a sistemului dupa entitatile identificate (ce inglobeaza toata functionalitatea obiectului).

1.2.4. Descrierea sistemului de calcul

Definiție. Masina de calcul care executa secvential programe scrise in limbajul masinii respective, stocate in memorie, in interactiune cu mediul extern.

Un program = solutie algoritmica a unei probleme scrisa intr-un limbaj, numit limbaj de programare.

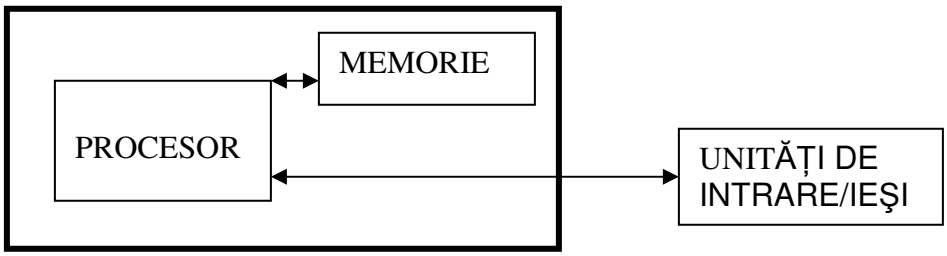
Un **algoritm**=solutie secventiala a unei probleme.

Limbajul de programare≠limbajul masina.

Limbajul masina este limbajul executat de masina.

Limbajul de programare este translatat in limbaj masina pentru executie.

Sunt doua forme de executie:
- **compilare si executie**
- **interpretare** (masina virtuala care interpreteaza si executa programul)



Elemente componente:
- de **procesare** (prelucrare)
- de **memorare**
- de **comunicatie**
  - **cu mediul extern**
  - **suport de comunicație**

Program = succesiune de instructiuni ce implementeaza un algoritm.

Din punct de vedere perceptiei, sistemul de calcul este impartit in doua mari parti:

- partea **hardware** – reprezentata de circuitele electronice, placi, cabluri, memorii, etc. care reprezinta echipamentul propriu-zis de calcul si care sunt tangibile.
- partea **software** – reprezentata prin programe care implementeaza algoritmi si reprezinta idei abstracte.

Diferenta dintre hardware si software pina nu demult a fost evidenta, cu timpul insa ele au devenit logic echivalente. Ambele putind realiza aceleasi functii, iar alegerea implementarii facindu-se dupa criterii pret/performanta.

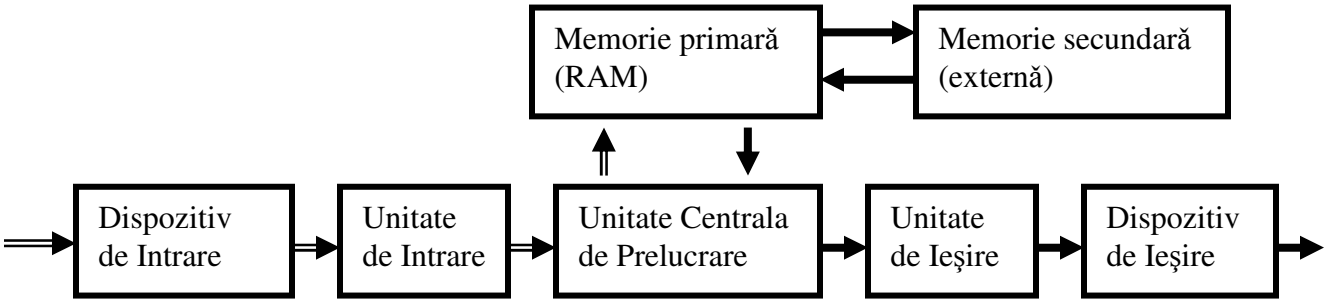
1.2.5. Partile componente ale sistemului de calcul.

Fluxul informației

Pentru a intelege functionarea calculatorului vom introduce notiunea de **informatie** care, furnizata de utilizator sau mediu, este convertita în format binar, intern, prelucrat de sistemul de calcul (***date***).

Adoptarea reprezentarii binare a fost impusa de utilizarea in constructia calculatoarelor a dispozitivelor cu doua stari stabile, notate conventional cu 0 si 1. Unitatea de masura pentru numerele binare este **bit**-ul (**bi**nary digi**t**).

Vom face in continuare o prezentare succinta a componentelor sistemelor de calcul cu referire directa la calculatoare. Numeroasele componente ale unui sistem de calcul pot fi grupate in unitati avind functii mai complexe bine precizate. In figura urmatoare denumirea fiecarei unitati indica functia ei, iar sagetile de legatura arata modul de transmitere a informatiei de la una la alta.



Informatia, furnizata de mediul extern (utilizator), este preluată de **dispozitivul de intrare**, codificată (convertită în format binar) şi transmisă **unității de intrare** care realizează interfața cu unitatea de procesare (unitatea centrală de procesare = UCP) .
Exemple de dispozitive de intrare: tastatura, mouse, scanner, MODEM, etc. Astfel, la tastatura, apasarea unei taste produce codului binar corespunzator al tastei apăsate. Scannerul preia o imagine şi o transformă într-o succesiune de coduri binare. MODEM-ul preia datele transmise de la distanță.
Unitatea de intrare realizează interfața cu UCP a.î. dispozitivele de intrare pot fi diferite.
Informatia este inregistrata si pastrata în **memoria primară**. De aici ea poate fi transmisa ulterior altor unitati functionale. Informatia este supusa prelucrarii in **UCP**. UCP este formată din **unitatea de calcul** şi **unitatea de comandă**. Unitatea de calcul efectueaza operatii simple, aritmetice si logice, asupra unor operanzi din memorie, inregistrind rezultatele tot in memorie. **Unitatea de comanda** are ca rol coordonarea functionarii celorlalte unitati, pe baza unor instructiuni sau comenzi.

Informatia care nu este prelucrata la un moment dat poate fi pastrata in unitati de **memorie externa** (de obicei discuri magnetice), mai lente decit **memoria interna** (operativa) dar cu

o capacitate mai mare. Informatia poate fi transmisa, daca este cazul,de la o memorie la alta.

Din punct de vedere al localizării, memoria sistemului de calcul este împărțită în:

- memorie internă
- memorie externă

Memoria internă este formată din:

- **RAM (Read Only Memory) –** volatila (îşi pierde conținutul dacă nu este alimentată cu curent electric) = memoria primară a sistemului de calcul (indispensabilă)
- **PROM (Programmable Read Only Memory)** - nevolatila = memorie care poate fi doar citită şi care cuprinde programe ale fabricanților sistemului de calcul
- **CMOS (Complementar Metal Oxid Siliciu)** – memorie asemănătoare memoriei RAM, de capacitate foarte mică, alimentată permanent de la o baterie, care păstrează setările de configurare a sistemului de calcul.

Memoria externa **- discuri magnetice şi optice.**

Rezultatele prelucrarilor sint transmise utilizatorului prin **unitatea de iesire** către **dispozitivul de ieşire**. Dispozitivul de ieşire realizeaza conversia datelor din format binar în formatul necesar reprezentarii informatiei. Exemple de dispozitive de ieşire : monitor, imprimnată, MODEM, plotter, etc. De exemplu, o imprimanta converteste codurile binare ale caracterelor in formatul tiparit. Similar, un monitor (display) transforma reprezentarile binare ale informatiei in formatul afişat.

1.2.6. Părți hardware principale componente :

- UCP (Unitatea Centrală de Procesare)
- Memoria
- Subsistemul de I/E
- Suportul de comunicație

UCP are rolul de a prelucra programul alcătuit din instrucțiuni şi de a controla activitatea întregului sistem.

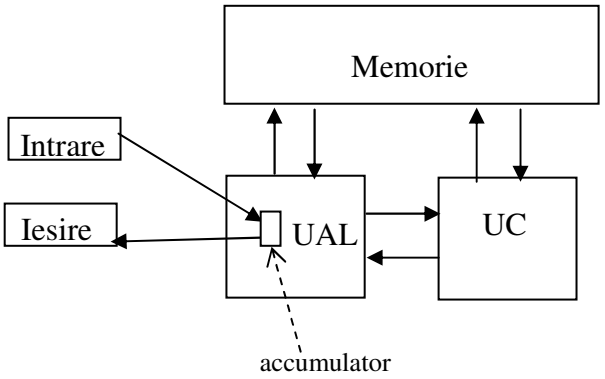
Memoria este cea în care se stochează informația în format binar.

Subsistemul de I/E realizează interfața cu mediul exterior.

Suportul de comunicație reprezintă infrastructura de comunicație necesară transmiterii datelor.

1.3. MODELUL VON NEUMANN

Modelul von Neumann a fost specificat de von Neumann in 1946 şi cuprinde elementele pricipale componente ale unui sistem de calcul şi interacțiunea între ele.
EI are la baza patru componente: *memoria*, *unitatea aritmetica si logica* cu un registru intern special numit accumulator, *unitatea de comanda* si *echipamentele de intrare si de iesire*.
Arhitectura masinii von Neumann este cea din figura urmatoare:



În această arhitectură programele sunt memorate şi ea stă la baza calculatoarelor moderne cu un procesor.

Unitățile componente se numesc şi **unități funcționale** deoarece fiecare are o funcționalitate specifică în sistem.
Unitatea centrală (UC) are rolul de a prelucra programul alcătuit din instrucțiuni şi de a controla activitatea întregului sistem. Instrucțiunea este memorată ca secventa de biți în memorie.
UC este alcătuită din:

- unitatea de comandă
- unitatea aritmetică şi logică (UAL) sau unitatea de calcul
- regiştrii

Unitatea de comandă decodifică instrucțiunile, le interpretează şi comandă operațiile de executat.
Unitatea de calcul execută operațiile aritmetice şi logice.
Regiştrii sunt folosiți pentru stocarea temporară a datelor de prelucrat. In modelul von Neumann este specificat doar registrul acumulator. Ulterior setul de regiştri a fost îmbogățit cu mai multe tipuri, în functie de datele stocate (adrese, date, stare, etc.).

Memoria este cea în care se stochează informația în format binar. De aceea suportul de memorie trebuie să asigure două stări stabile distincte. Ea este compusă dintr-un şir de **locații de memorie,** iar accesul la ele se face prin **adrese**. Locația de memorie are

dimensiunea de un **octet.** Un octet fiind format din 8 biți. Un **bit** putând stoca o valoare binară (0 sau 1). Numărul locațiilor de memorie formează **capacitatea de stocare** a memoriei. În memorie sunt stocate atât date cât şi programe.
Operațiile care se execută cu memoria sunt: de citire şi de scriere

Într-un sistem de calcul există mai multe tipuri de memorie. Ele sunt dispuse ierarhic în sistem în funcție de timpul de acces şi de prețul acestora. Astfel pe nivelul inferior se află **memoriile de masă** reprezentate prin discuri magnetice, CD-uri, având timp mare de acces, preț mic şi corespunzător capacitate mare. În memoriile de masă se stochează informația persistentă în timp. Pe urmatorul nivel se află **memoria primară (RAM=Random Acces Memory).** Această memorie este indispensabilă în sistemul de calcul, fiind cea cu care se execută programele. Timpul de acces este mic, pretul mai mare şi capacitatea de stocare mult mai mică decât cea a memoriei de masă. Caracteristica principală memoriei primare este **volatilitatea**, adică pierderea informațiilei la căderea tensiunii. Cu această memorie se efectuaează atât operații de scriere cât şi operații de citire.
Următorul nivel îl constituie cel al memoriilor de tip **cache**. Acestea sunt memorii foarte rapide, mai scumpe, folosite pentru accelerarea vitezei de lucru a calculatorului.
Pe ultimul nivel se alfă **regiştrii** care sunt cei mai rapizi aflându-se direct conectați la unitățile de procesare. Capacitatea lor de stocare este foarte mică, iar numărul lor este limitat de dimensiunea microprocesorului.
Alte tipuri de memorii aflate în sistem sunt:

- memoria **ROM (Read Only Memory),** care stocheaza programele furnizate de fabricantul calculatorului. Singura operație efectuată cu acest tip de memoruie este citirea ei.
- memoria **CMOS** (numele este dat de tehnologia de realizare a acesteia). În această memorie sunt păstrate informații de configurare a calculatorului. Cu ea se realizează operații de citire şi scriere şi este alimentată de la o baterie.

Unitatea de intrare este componenta care asigură funcția de preluare a informațiilor de intrare. Acestea sunt **citite** de la un **dispozitiv de intrare**.
Unitatea de ieşire este componenta care asigură funcția de furnizare a informațiilor la ieşire. Acestea sunt **scrise** şi transmise unui **dispozitiv de ieşire**.
Pentru sincronizarea unităților funcționale componente, există între ele **interfețe** în care se află **buffere (elemente temporare de memorare**).
Înformatia este transmisă în sistem pe căi electrice numite **magistrale**. În funcție de tipul de informație care circulă prin ele, ele se clasifică în: **magistrala de adrese, magistrala de date, magistrala de control.** Pentru a se obține flexibilitate în interconectarea diverselor componente ale sistemului de calcul, magistralele sunt standardizate.

1.4. ARHITECTURA STRATIFICATĂ A SISTEMULUI DE CALCUL

Calculatorul numeric este o masina care poate rezolva probleme prin executia unor instructiuni. Solutia problemei, descrisa prin secventa de instructiuni, se numeste **program**. Instructiunile sunt scrise dupa un anumit limbaj. Exista mai multe tipuri de limbaje care pot fi folosite pentru programarea calculatorului si deasemenea mai multe niveluri la care se poate privi sistemul de calcul.
In cartea sa “Organizarea structurata a calculatoarelor”, Andrew S. Tanenbaumn descrie sistemul de calcul ca o ierarhie de niveluri abstracte, fiecare abstractizare bazandu-se pe cea inferioara in realizarea functiilor sale fara a cunoaste insa in detaliu continutul

acesteia. Calculatorul privit la nivelul respectiv este considerat o **masina virtuala**, M_i, iar limbajul de programare pentru respectivul nivel **limbaj masina**, L_i. La nivelul cel mai inferior se afla calculatorul real care executa propriu-zis instructiunile sale masina, L_0. Nivelul superior comunica instructiunile sale nivelului inferior, care trebuie să le execute (evident prin instructiunile de nivel inferior). Instructiunile de nivel superior sunt ori **translatate** in instructiuni de nivel inferior (o intructiune L_1 in mai multe instructiuni L_0) formindu-se astfel un program in L_0 si executat apoi de M_0, ori **interpretate** prin translatarea si executia pe rind a instructiunilor L_1.

NIVELUL LIMBAJULUI DE NIVEL INALT
NIVELUL LIMBAJULUI DE ASAMBLARE
NIVELUL SISTEMULUI DE OPERARE
NIVELUL ARHITECTURII SETULUI DE INSTRUCTIUNI
NIVELUL MICROARHITECTURII
NIVELUL LOGIC DIGITAL

Calculatoarele moderne sunt formate din doua sau mai multe niveluri. Exista masini cu sase niveluri. In fig. Nivelul 0 corespunde structurii hardware a masinii, nivelul 1 corespunde microarhitecturii, nivelul 2 arhitecturii setului de instructiuni, nivelul 3 sistemului de operare, nivelul 4 limbajului de asamblare, nivelul 5 limbajului orientat pe problema.

Nivelul 0 numit si **nivel logic digital** cuprinde si **nivelul echipamentelor** (care se afla la cel mai scazut nivel si in care utilizatorul poate vedea tranzistoarele). La nivelul digital se afla obiectele numite **porti(gates**) care sunt construite din tranzistoare, dar au intrari si iesiri digitale (semnale ce reprezinta "0" sau "1"). Portile pot fi combinate pentru a forma o memorie de un bit care poate stoca "0" sau "1", iar memoriile de 1 bit pot fi combinate pentru a forma grupuri de 16, 32 sau 64 biti, numite **registre**.

La **nivelul microarhitecturii** se afla registre, o memorie locala si un circuit numit **UAL (Arithmetic and Logic Unit)** care poate executa operatii simple aritmetice si logice. UAL este o unitate combinationala cu doua intrari si o iesire. Aici se poate distinge o cale de date, prin care circula datele, una de adrese si una de control si stare. La unele calculatoare operatiile caii de date sunt controlate prin *microprogram*, iar la altele direct prin hardware. Microprogramul este un interpretor al instructiunilor nivelului superior.

Nivelul setului de instructiuni, numit si **ISA** (**Instruction Set Architecture**) reprezinta primul nivel programabil de catre utilizator. Aici gasim instructiunile masina furnizate de producator, care la rindul lor sunt interpretate sau executate prin hardware-ul de la nivelul inferior.

Nivelul urmator, al **sistemului de operare** este hibrid, adica se intilnesc si instructiuni ISA si instructiuni noi ale sistemului de operare care sunt interpretate de sistemul de operare sau direct de microprogram.

Primele trei nivele nu sunt utilizate de programatorul obisnuit, ci de **programatorii de sistem** care realizeaza sau intretin interpretoarele sau translatoarele pentru masina virtuala. De la nivelul 4 in sus masina virtuala este folosita de programatorii de aplicatie. Nivelurile 2 si 3 sunt de obicei interpretate, iar de la 4 in sus translatate, desi exista si exceptii. Alta deosebire este ca limbajele primelor trei niveluri sunt numerice (i.e. sunt alcatuite din siruri de numere). La nivelul 4 limbajul de programare se numeste **de asamblare** si reprezinta o scriere simbolica pentru nivelul inferior. Programul care interpreteaza limbajul de asmblare se numeste **asamblor**.

Limbajele de la nivelul 5 se numesc **limbaje de nivel inalt** (**HLL = High Level Language**) si sunt realizate pentru programatorul de aplicatii. Aici gasim: C, C++, Pascal, Prolog, Java, LISP, etc. Aceste limbaje sunt traduse prin translatoare numite **compilatoare** (exista si exceptii: Java este interpretat). Tot la acest nivel se afla si interpretoarele pentru aplicatii particulare anumitor domenii (matematica, chimie, etc.).

1.5. GENERATII DE CALCULATOARE

Din punct de vedere al tehnologiei de realizare a calculatoarelor, acestea se impart in generatii de calculatoare.
Generatia 0 – reprezentata de **calculatoarele mecanice** – 1942-1945.
Blaise Pascal a realizat prima masina de calcul functionala, complet mecanica, din roti dintate si actionata de manivela in 1942.
Urmatoarea masina, realizata in secolul al XIX-lea de Charles Babbage, s-a numit **masina de calcul a diferentelor** (**difference engine**) si era specializata pentru un singur tip de calcule pentru navigatia marina. Ea efectua numai adunari si scaderi. Tot Babbage a realizat si **masina analitica** (**analitical engine**) cu functionalitate mai mare ca cea precedenta, programabila intr-un limbaj simplu de programare de **Ada** Aususta Lovelace, fiica lordului Byron.
In 1930 germanul Konrad Zuse realizeaza o serie de masini de calcul folosind releele eletromagnetice.
In SUA in anii celui de-al doilea razboi mondial, John Atanasoff si George Stibbitz proiecteaza masini de calcul automate, iar Aiken a realizat masina Mark I din relee electromagnetice, avind o banda de hirtie perforata la intrare, 72 de cuvinte si 6 secunde pe instructiune.

Generatia 1 de calculatoare – având la baza **tehnologia tuburilor electronice** – 1943-1953
Primul calculator numeric electronic din lume a fost considerat COLOSSEUM, 1943, construit de guvernul britanic pentru decodificarea (prin dispozitive ENIGMA) mesajelor germane in timpul celui de-al doilea razboi mondial. Alan Turing a contribuit la proiectarea acestuia.
Celebrul calculator **ENIAC** (**Electronic Numerical Integrator And Computer**) a fost realizat de John Mauchley si J. Presper Eckert. El avea 18 000 de tuburi electronice, 1500 de relee, 20 de registre a cite 10 digiti si cântarea 30 de tone. Din pacate nu a fost functional. EDSAC s-a numit primul calculator numeric functional si a fost realizat la Universitatea Cambridge de Maurice Wilkes, iar alt reprezentant demn de specificare este EDVAC (Electronic Discrete Variable Automatic Computer).
Bazele aritecturii calculatoarelor de astazi au fost puse de *John von Neumann* care a realizat **masina IAS**. El a propus inlocuirea aritmeticii zecimale seriale cu aritmetica

binara paralela. **Masina von Neumann** a fost realizata de Wikes si s-a numit EDSAC avind integrat un program memorat.
IMB produce masinile 701, apoi 704 si 709 intre anii 1953-1958.

Generatia 2 de calculatoare – avind la baza **tehnologia tranzistoarelor** – 1955-1965
Tranzistorul a fost inventat în 1948 la Bell Laboratories, iar printre inventatori a fost William Shockley.
Primul calculator cu tranzistoare s-a realizat la MIT şi s-a numit TX-0 (Transistorized eXperimental computer).
În 1961 a apărut calculatorul PDP-1, realizat de firma DEC (Digital Equipment Corporation) cu 4K cuvinte pe 18 biţi şi cu timpul ciclului maşină de 5 μsec. Calculatorul PDP8 care a urmat a adus nou magistrala comună plimbata pe la toate componentele, numita omnibus.
Magistrala=colecţie de fire paralele folosite pentru conectarea componentelor unui calculator.
In 1964, **CDC** (Control Data Corporation) a realizat masina **6600**, mult mai rapidă decât celelalte existente la ora respectivă, având UCP (Unitatea Centrală de Procesare) masiv paralelă. Seymour Cray fiind unul dintre proiectanţi care trebuie menţionat.
Firma IBM a realizat în această perioadă calculatoarele 7094 şi 1401, unul pentru calcule ştiinţifice, altul pentru gestiunea afacerilor.

Generaţia 3 - având la bază **tehnologia circuitelor integrate** – 1965-1980
Circuitul integrat a fost inventat de *Robert Noyce* în 1958 şi a dus la miniaturizarea echipamentelor prin integrarea a zeci de tranzistoare pe un cip.
Firma IBM a realizat System/360 atât pentru calcule ştiinţifice cât şi comerciale. În plus acest calculator aduce nouă **multiprogramarea** (**multiprogramming**). Această tehnică permite existenţa în memorie a mai multor programe, cele care realizează transferuri I/E (intrare/ieşire) lăsând UCP-ul celor care realizează calcule.
Firma DEC realizează în această perioadă foarte popularele calculatoare PDP8 şi PDP11.

Generaţia 4 - având la bază **tehnologia circuitelor integrate pe scară foarte largă** – 1980-?
Acestă generaţie este cea a calculatoarelor personale.
Primele calculatoare au la bază microprocesorul I8080 şi sistemul de operare CP/M (scris de Gary Kildall) şi sunt realizate din componente asamblate la cerere.
Calculatorul **Apple** realizat de Steve Jobs şi Steve Wozniac apare la începutul perioadei devenind foarte popular.
IMB lansează în 1981 primul calculator personal din componente comerciale, numit IBM PC, cu sistemul de operare MS-DOS, realizat de firma Microsoft Corporation.
Structurile RISC apar la mijlocul anilor 1980 cu performanţe crescute faţa de cele oferite de direcţia CISC.

1.6. LEGI EMPIRICE

Legea lui Moore (legea hardware-ului)
În anul 1965, Gordon Moore , fondator al companiei Intel, obsevând că numărul de tranzistoare creştea constant, a prezis că numărul acestora se va dubla anual. Aceasta a devenit **legea lui Moore**, exprimată ca dublarea numărului de tranzistoare la fiecare

18 luni. Evident acest progres tehnologic a dus la creşterea performanţelor sistemelor şi la scăderea preţurilor.

Legea software-ului

Enunţată de **Nathan Myhrvold** spune că "software-ul este ca un gaz, crescându-şi volumul astfel încât să ocupe tot spaţiul pe care îl are la dispoziţie". Această lege indică faptul că resursele hard disponibile sunt imediat consumate de către soft, chiar mai mult existând o cerere permanentă de resurse.

1.7. TIPURI DE CALCULATOARE

Există două direcţii importante în dezvoltarea calculatoarelor:

- **CISC** (Complex Instruction Set Computers) corespunzătoare calculatoarelor realizate cu microprocesoare cu arhitectură CISC.
- **RISC** (Reduced Instruction Set Computers) corespunzătoare calculatoarelor realizate cu microprocesoare RISC, reprezentativ fiind microprocesorul SPARC realizat de firma Sun.

In paralel sunt dezvoltate direcţii alternative:

- Calculatoare paralele. Exemplu: reprezentativ este **MIPS** (Milions of Instruction Per Second) realizat la Universitatea Stanford USA, cu arhitectură mai specială, paralela.
- Calculatoare orientate către limbaj: direcţie nouă de dezvoltare o constituie cipurile **JVM** (Java Virtual Machine).

Exemple de tipurile de calculatoare sunt:

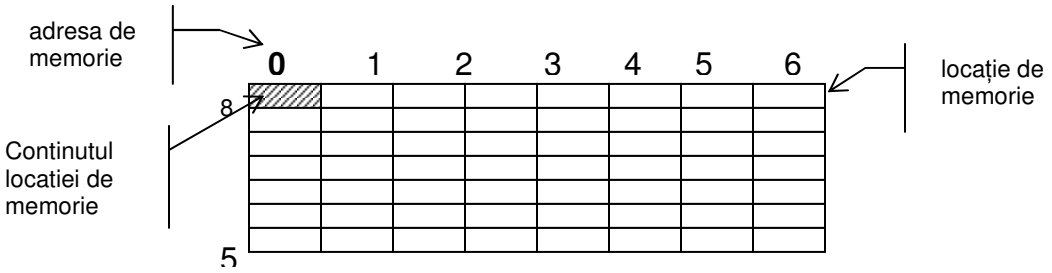
- **Calculatoare personale** –ele se referă la calculatoarele de birou şi la agendele de lucru. Ele sunt monoprocesor şi se numesc PC-ri (daca microprocesorul este CISC) sau staţii de lucru (dacă procesorul este RISC). Puterea lor de calcul creşte pe măsura evoluţiei tehnologice. Pot fi echipate cu MODEM-uri pentru transmisia la distanţă.
- **Server-e** – Ele se referă la calculatoarele cu putere mai mare din reţea pe care se află instalat software-ul corespunzător, deservind staţiile de lucru.
- **Mulţime de staţii de lucru** – numite şi **Networks of Workstations (NOW)**, sau **Clusters of Workstations (COW)** – sunt alcătuite din mai multe staţii de lucru legate prin reţele de mare viteză şi având un software distribuit pentru soluţionarea împreună a unor probleme specifice unui domeniu.
- **Calculatoarele mari** - specifice sistemelor mari cu capacitate foarte mare de stocare (de ordinul teraoctetilor, 1Toct.=10^{12} oct.).
- **Supercalculatoarele** - cu UCP foarte rapide, resurse mari (memorie) şi interconectări rapide folosite pentru calcule foarte complicate ştiinţifice.

Toate aceste calculatoare au unitatea centrală de prelucrare (CPU = Central Processing Unit) integrată pe un chip, numit microprocesor.

2. MEMORIA SISTEMULUI DE CALCUL

2.1.Organizarea memoriei

Memoria stochează informația în format binar.
Ea este compusă din *locații de memorie* care păstrează datele. Fiecare locație este referită printr-o *adresă*. Se poate imagina memoria ca un fişet cu multe sertare. Fiecare sertar reprezentând o locație de memorie cu o adresă unică şi care conține diferite obiecte. Conținutul sertarului se poate modifica prin obiectele care le conține. La fel şi conținutul locației de memorie.
Dimensiunea locației de memorie este de un octet.



2.2. Reprezentarea informației

Deoarece orice sistem de calcul lucrează cu informația în sistem binar, adresele si datele stocate sunt reprezentate prin simple *numere binare*.

Unități de măsură a informației:

- bitul
- octetul (byte)
- cuvântul (word)
- dublucuvântul (doubleword)
- multiplii:
  - 1 Koctet = 2^{10} octeți
  - 1Moctet = 2^{20} octeți
  - 1Goctet = 2^{30} octeți
  - 1Toctet = 2^{40} octeți

2.3. Caracteristicile memoriei:

1. *capacitatea memoriei* = numărul de locații de memorie
2. *timpul de acces* = timp de obținere a valorii stocate în locația de memorie de la accesarea ei.
3. *rata de transfer* = număr de locații transferate în unitatea de timp
4. *operațiile* efectuate asupra ei:
   - citire = redarea datelor stocate în memorie
   - scriere = stocarea datelor în memorie.

2.4. Tipuri de memorie

Din punct de vedere al localizării, există două tipuri de memorie în sistem:

- memoria internă – accesată direct de microprocesor
- memoria externă – stochează permanent datele care sunt transferate în memoria RAM pentru procesare

2.5. Memoria internă

Memoria internă este compusă din:

- memoria *RAM (Random Access Memory)* – cu acces aleator la date, care realizează ambele tipuri de operații (citire, scriere)
- memoria *ROM (Read Only Memory)* – realizează numai operația de citire (conținutul nu poate fi alterat).
- Memorie CMOS – alimentată permanent de la o baterie şi care stochează informații de configurare (setup) ale sistemului de calcul.

Ierarhia memoriei

.....

2.5.1. Memoria RAM

Există două tipuri de bază (în funcție de implementarea ei):

- memorie *statică (SRAM)*

Memorie realizată din CBB (circuite basculante bistabile) care pentru anumite semnale date la intrare si în funcția de starea anterioară a circuitului, furnizează o ieşire de 0 sau 1.

Are timp mic de acces, deci este foarte rapidă.

Păstrează informația cât timp este alimentată cu tensiune.

- memorie *dinamică (DRAM)*

Realizată din condensatori, încărcați cu o sarcină electrică sau descărcați, corespunzător lui 1 respectiv 0 logic.

Deoarece condensatorii se descarcă în timp trebuie reîncărcați periodic. Operația de reîncărcare se numeşte *refresh de memorie*.

Timpul de acces este mai mare decât la SRAM.

Pentru memoria RAM mai există următoarele implementări:

- EDO RAM – Extended Data Out RAM – care sealizează simultan citirea şi scrierea memoriei
- ERAM – Enhanced RAM – memorii DRAM care implementează şi paginarea memoriei = FPM (?) Fast Page Mode – memorii cu paginare rapidă.
- S-RAM – Synchronous RAM - sincronizate separat pe baza unui semnal de tact şi cu un cache integrat
- CDRAM – Cached RAM – idem S-RAM, dar cu cache-ul mai mare
- R-RAM – RAMBus RAM – memorie cu magistrale interne rapide.

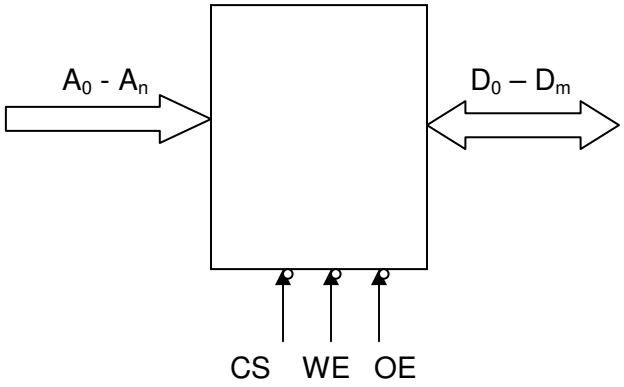
Cipuri de memorie

Pentru a obține o dimensiune de memorie se pot folosi diferite cipuri de memorie (de 1, 4, 8, 16, 32 biți). Cipurile grupate formează plăcuțe de tip:

- SIPP (Single Inline Pin Package) – cu conector în formă de pini
- SIMM (Single Inline Memory Modules) – cu conector în formă de placă
- DIMM (Dual Inline Memory Modules)

Acestea se introduc în soclul de memorie aflat pe placa de bază.

Un cip de memorie este un circuit integrat cu pini de intrare, ieşire şi intrare/ieşire.



Unde:

- A_0 – A_n reprezintă biții de adrese
- D_0 – D_m reprezintă biții de date
- CS este pinul de selecțe a circuitului (ChipSelect)
- WE este pinul de activare a scrierii (WriteEnable)
- OE este pinul de activare a ieşirii (OutputEnable)

2.5.2. Memoria cache

Memoria de tip RAM este mai lenta decat viteza de lucru a microprocesorului. In consecinta datele solicitate de microprocesor vin mai lent, iar microprocesorul trebuie sa introduca stari de asteptare (wait states) pentru sincronizarea operatiilor. Memoria cache este introdusa ca o memorie tampon, mai rapida si mai aproape de microprocesor (timpul necesar parcurgerii traseului pana la memorie fiind mult mai scurt).

Memoria cache este divizata fizic in doua nivele:

- *memoria cache primara* sau *interna* sau memoria cache pe circuit, plasata in chiar chipul microprocesorului, dar de dimensiuni mai mici
- *memoria cache secundara* sau *externa* sau memoria chache pe placa, plasata in exteriorul microprocesorului si de dimensiuni mai mari

Activitatea memoriei cache este controlata de un un *controller de memorie cache* care gestioneaza adresele si operatiile de citire/scriere efectuate de memorie.

La nivelul memoriei cache interne se realizeaza o distinctie intre tipurile de date stocate in memorie. In consecintia implementarile sunt diferite pentru memoria cache pentru cod, memoria cache pentru date si memoria cache pentru gestiunea memoriei. Memoria cache pentru cod are o implementare de conducta deoarece programele se executa secvential. In conducta este incarcata o secventa de instructiuni care este cel mai probabil sa se execute. Conducta se goleste la aparitia instructiunilor de salt, dar se reumple cu instructiunile de la adresa de salt. Celelalte memorii cache au implementari asociative deoarece informatia nu este atat de predictibila ca a codului. Microprocesoarele mai noi dispun de doua nivele de cache intern.

Memoria cache externa se afla pe placa de baza a sistemului si este formata din cipuri de memorie foarte rapide (de tip SDRAM) cu timpi de acces foarte mici. La acest nivel

nu se realizeza o distinctie intre tipurile de date manipulate, iar implementarea pentru aceasta memorie este asociativia.

Reinnoirea continutului memoriilor cache pentru stergerea datelor curente si aducerea de date noi se face folosind metode de tipul:

- metoda RW (Random Write) sau de rescriere aleatoare .Datele sunt rescrise aleator ,fara a folosi un anumit criteriu sau algoritm care sa determine care bloc de date va fi scris.
- Metoda FIFO – care implementeaza mecanismul First In First Out .
- Metoda LRU (Last Recently Used) in care se realizeaza o statistica a utilizarii blocurilor de date ,iar cele mai utilizate sunt pastrate in memorie. Aceasta este si cea mai folosita metoda.

Memoria cache este conectata la microprocesor si RAM in mod serial sau paralel.
Conectarea seriala (look trough) presupune ca dialogul microprocesor – RAM sa se realizeze prin intermediul cache-ului .Dezavantajul acestei conectari este ca cererile procesorului sunt intarziate de cache .Avantajul metodei este ca magistrala sistem este eliberata in cazul in care datele se gasesc in cache.
Conectarea paralela (look aside) presupune adresarea paralela atat memoriei cache cat si memoriei RAM. In cazul in care datele sunt gasite in cache cautarea din RAM este abandonata .Avantajul conectarii este viteza ,evitandu-se cautarea secventiala ,intai in cache apoi in RAM .Dezavantajul este ocuparea magistralei de memorie care intarzie accesul altor periferice (dispozitive de intrare-iesire) si activitatile in regim de multiprocesoare.

Modul de lucru pentru memoria cache este urmatorul: microprocesorul cauta datele necesare in cache-ul intern, daca nu le gaseste (cache miss) le cauta in cache-ul extern, iar daca nu sunt nici in acesta le cauta in memoria RAM.

Depunerea in memorie a rezultatelor operatiilor se realizeaza folosind una din metodele :

- metoda Write Through prin care se utilizeaza traseul invers al citirii datelor, adica acestea se depun intai in memoria cache interna apoi in cacheul extern si in final in memoria RAM
- metoda Write Back in care scrierea se face in cache, in cazul in care adresa de memorie la care se face scrierea exista in cache, si direct in RAM daca blocul nu mai exista in cache.
- Metoda Posted Write prin care in memoria cache este rezervat un buffer in care se memoreaza blocuri de date pana ce magistrala de memorie devine disponibila , cand se transfera continutul in RAM.

Celulele cache pot fi :

- *asincrone*, la care operatia de citirea unui cuvant se face in doua etape:se depune adresa apoi se transfera datele (pentru fiecare cuvant)
- *sincrone* pentru care accesul se face in mod burst (adica se depune o singura adresa, iar transferul se face pentru patru cuvinte simultan)
- *pipelined burst cache* care este prevazut cu registre speciale in care datele citite sunt stocate temporar.Astfel este posibila o noua adresare, practic in acelasi timp cu preluarea datelor de catre microprocesor.

2.6. Memoria externă

Memoria externă reprezintă suportul pentru stocarea permanentă a datelor.
Caracteristici:

- caracteristicile generale ale memoriei
- *densitatea de înregistrare a datelor* = număr de biți/unitate de stocare.

Medii de stocare a datelor

1. Magnetice:
   - Harddiscul (HD)
   - Floppy discul (FD)
   - Harddiscul portabil (HPD)
   - Banda magnetică (streamer)
2. Optice:
   - Discul CD-ROM
3. Magneto-optice:
   - Discul magneto-optic
4. Memoria Flash

Banda magnetică

- informația stocată în format binar similar procesului Hi/Fi
- există două tipuri de benzi magnetice:
  - cu înregistrare analogica (capacități de 50MB – 1GB)
  - cu înregistrare digitală (DAT – Digital Audio Tape) – capacități de până la 5GB.şi densitate mare de înregistrare a datelor şi mult mai performante beneficieând de avantajele prelucrărilor digitate. Sunt stocate şi informații de control pentru securizarea datelor.
- accesul la informație se face *secvențial*
- informația este organizată în cadre, înregistrări, piste.
- preț mic => capacitate mare => ideală pentru stocări masive de date
- timp mare de acces

Mai utilizată este banda *Streamer*.

Discul flexibil (Floppy disk)

- Construcție: - disc flexibil din plastic pe care este depus un strat subțire de material magnetic (oxid de crom). Dispun de capete de citire/scriere pentru fiecare față a discului.
- înregistrarea informației se face prin magnetizare/demagnetizare.
- Capacitate mică de stocare a datelor
- Timp mare de acces.
- Rată mică de transfer.
- Unitatea de disc cuprinde un controller de disc care dă funcționalitate prin implementarea interfeței cu unitățile componente şi a operațiilor asupra discului.
- Organizarea informației pe disc în sectoare, piste (numerotate începând cu 0 din centrul discului).
- Pentru păstrarea informației pe disc trebuie realizată opearția de formatare.
- Acces direct la blocurile de memorie
- Accces simultan la mai multe fişiere

Harddiscul

- Numit şi disc *dur,amovibil, fix* sau *Winchester*, este cel mai utilizat mediu de stocare în sistemele actuale.
- Construcție: mai multe discuri magnetice suprapuse, prinse pe acelaşi ax, înregistrate pe ambele fețe, capete de citire/scriere pentru fiecare față.

In funcție de mobilitatea capetelor există: HD cu *capete fixe* şi HD cu *capete mobile*.

- Capacitate mare de stocare a datelor
- Timp de acces mai bun decât la floppy
- Rată mare de transfer a datelor
- Densitate mare înregistrare
- Organizarea informației pe disc:
    - *piste* – zone circulare concentrice ape unui disc. Numerotarea pistelor prin *interțesere (interleaving*)
    - *Sectoare* – reprezintă unități de transfer a informației pe disc şi de diviziune a pistelor (o pistă formată din mai multe sectoare). Un sector este format dintr-un număr de octeți (putere a lui 2). La DOS, 512 octeti.
    - *Clustere* – organizare logică a mai multor sectoare şi formând unități de alocare.
    - *Cilindri* – reprezintă pistele active. Un cilindru este format din pistele cu aceeaşi rază.
- Acces direct la blocurile de memorie: pentru a ajunge la o înregistrare trebuie indicat cilindrul, pista, sectorul şi adresa de început în sector.
- Accces simultan la mai multe fişiere
- Discul trebuie pregătit pentru păstrarea informației prin formatare. Aceasta are două etape:
    - *Formatarea primară* (de nivel scăzut-low level) – stabilirea structurilor de piste circulare, concentrice şi a sectoarelor pe disc.
    - *Formatarea logică* – ce cuprinde partiționarea discului corespunzător căreia discul este împărțit în mai multe unități logice numite *partiții*.

Discul optic

- Densitate foarte mare de înregistrare pe suprafață mică
- Construcție: material policarbonat, foarte dur şi rezistent pe care se depune un strat fin de aluminiu reflectorizant.
- Tehnologie laser pentru înregistrare/redare. Scrierea se face printr-o rază foarte dură care arde adâncituri (pits) pe spirală. Zonele nealterate ale spiralei sunt numite lands. Un 1 logic este reprezentat printr-o succesiune pit/land, iar 0 logic prin succesiuni pit-pit sau land-land.
- Două tipuri în funcție de caracterul permanent al datelor stocate:
    - Discuri reversibile R/W
    - Discuri ireversibile RO
- Organizarea informației pe disc – o pistă în spirală cu parcurgere din interior la exterior. Pista este împărțită în sectoare cu dimensiunea de 2048 octeți.
- Capacitate mare de stocare
- Timp bun de acces
- Densitate mare de înregistrare.
- Exemplu: discurile CD-ROM

Discul magneto-optic

- operațiile de citire/scriere se fac prin procedee magnetice şi optice.

Memoria Flash

Memorie asemanatoare celei RAM cu consum redus de energie si care se cupleaza foarte rapid. Are forma unei cartele de credit (extraplata). Este accesata printr-o interata speciala.
Iscrierea se face ca la EPROM-uri.
Timp de acces foarte bun.

Alimentate de la o baterie.
Capacitate mica de stocare.
Pret foarte mare. Utilizate la laptopuri.

2.7. Codificarea informației

Informația, pentru a fi procesată, trebuie transformată în sistem binar. Acest lucru se realizează prin *codificare*.

Reprezentarea numerelor:

Numerele întregi sunt sunt reprezentate binar în mărime şi semn.

N=

Numerele negative sunt reprezentate în complement față de doi.

....

Numerele reale sunt reprezentate în virgulă mobilă.

....

Adresele sunt şi ele simple numere binare

Reprezentarea caracterelor

Există mai multe standarde de codificare a informației, dintre care cele mai utilizate sunt:

- ASCII – pentru caractere – cod pe 7/8 biți pentru reprezentarea caracterelor
- UNICODE – cod pe 16 biti (2 octeți pentru reprezentarea caracterelor)
- BCD – cod pe 4/8 biți pentru reprezentarea cifrelor zecimale

Ordinea de stocare a datelor în memorie

Pentru date mai mari de un octet există două posibilități de ordonare:

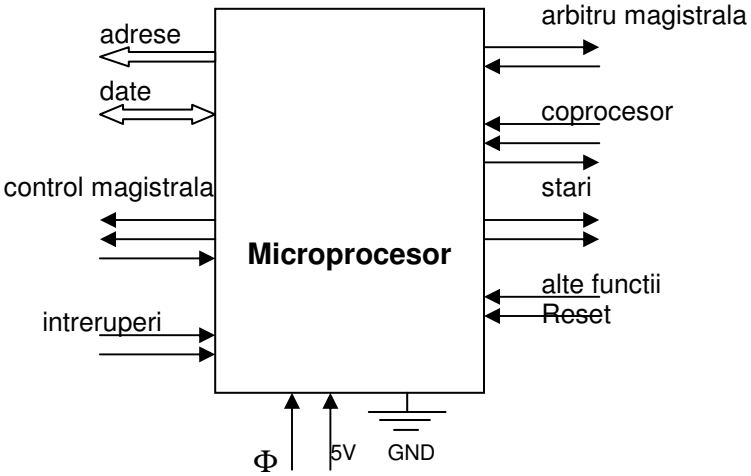
- *big endian* – în care octetul mai semnificativ este memorat întâi (la adresă mai mică)
- *litle endian* – în care octetul mai semnificativ este memorat la adresă mai mare.

Coduri detectoare/corectoare de erori

3. MICOPROCESORUL

3.1. Cipul microprosorului

Microprocesorul este “creierul” calculatorului. El este o unitatate centrală de procesare încorporată într-o singură pastilă de circuit integrat (un cip sau chip) care citeşte instrucțiunile unui program depus în memoria principală, le decodifică şi le execută secvențial (una după alta).



Cipurile microprocesor comunica cu lumea exterioara prin **pini**. In functie de directia pe care circula semnalele acestia se clasifica in:

- pini de intrare – prin care microprocesorul primeste semnale de la celelalte unitati functionale ale sistemului de calcu
- pini de iesire – prin care microprocesorul transmite semnale celorlalte unitati functionale
- pini de intrare/iesire – prin care microprocesorul poate primi si transmite semnale..

In functie de tipul datelor vehiculate prin pini avem:

- pini de adresa
- pini de date
- pini de control (magistrala, intreruperi, semnalizare, etc.)

pini de stare

- pini de alimentare si masa

3.2. Unități funcționale

Unitatile functionale componente ale microprocesoarelor actuale sunt:

CPU (Central Processing Unit) – unitatea centrală de prelucrare. Reprezintă unitatea de procesare (execuție) a instrucțiunilor, aritmetica numerelor întregi şi de coordonare a întregului sistem.

FPU (Floating Point Unit) – unitatea în virgulă mobilă, specializată în aritmetica numerelor reale reprezentate în format virgulă mobilă (standard IEEE 754)

MMU (Memory Management Unit) – unitatea de gestiune a memoriei, realizează automat gestiunea memoriei.

MMX (MultiMedia eXtension) – unitate multimedia specializată în prelucrări grafice

3.3. Unitatea centrală de prelucrare

CPU include următoarele unități funcționale:

- **UAL (Unitatea Aritmetica si Logica)** reprezinta o unitate combinațională cu două intrări şi o ieşire care executa operații aritmetice si logice.
- **UC (Unitatea de Comanda)** este unitatea functionala care programează execuția secvențială a tuturor operațiilor necesare efectuării instrucțiunilor, generând semnale de comandă pentru tot sistemul, dirijând fluxul de date, corelând viteza de lucru a unității centrale de prelucrare cu memoria, etc.. Activitatea unității de comandă este pilotată de un *semnal de ceas* a cărui frecvența este acum de ordinul sutelor de MHz.
- **Registrii** Aceştia reprezintă elemente de memorare în care se stochează temporar date sau adrese. Unii sunt folosiți pentru urmărirea execuției instrucțiunilor (registrul contor de program), alții sunt folosiți în calcule (regiştri cu scop general), altul păstrează starea programului în execuție (registru de stare), alții pentru calculul adreselor de memorie (regiştri de adresă). Aceştia reprezintă cea mai rapidă formă de memorie din sistem fiind direct conectați la UAL.

Efectuarea transferurilor de date şi comenzi intre unitatile functionale ale microprocesorului se face pe **magistrala internă de date** a microprocesorului.
Semnalele electrice prin care microprocesorul dă comenzi de execuție către memorie şi către celelalte componente din sistem se numesc **semnale de comandă**.
Semnalele electrice prin care microprocesorul culege informatii privind starea componentelor din sistem se numesc **semnale de stare**.

Lungimea (numărul de biți) regiştrilor interni se corelează de obicei cu *lățimea* (numărul de linii) ale magistralei de date. Aceasta e **măsura numărului de biți** ai microprocesorului. Microprocesoarele cu structurv fixă sunt de 8,16,32,64 biți. Lungimea de cuvânt a microcalculatoarelor realizate cu microprocesoare « bit slice » (felii de bit), a căror structură e flexibilă, va fi un multiplu întreg al numărului de biți ai unei felii.
Registrul de adrese, respectiv lățimea magistralei de adrese defineşte spațiul de memorie adresabil direct de microprocesor . O magistrală de adrese de 16 biti permite adresarea a 2^{16}=65536 celule distincte, iar 20 linii de adresă ne duc în lumea megaocteților: 2^{20}=1.048.576 celule adresabile.

Memoria adresata direct de microprocesor se poate impartii in memorie program si memorie de date. Memoria program contine instructiuni executabile de catre microprocesor, iar memoria de date date utilizate de instructiunile programului.
Datele utilizate in program pot fi adrese (de adresare) sau date stocate in memorie (adresate).

3.4. Caracteristicile microprocesorului :

- Frecventa ceasului
- Lațimea magistralei de date
- Lațimea magistrala de adrese
- Setul de instructiuni

- Arhitectura UCP

3.5. Istoric al evolutiei microprocesoarelor

3.5.1. Microprocesoare INTEL

1964 - Gordon Moore – enunt lege “numarul de tranzistori planari pe pastiala de siliciu se dubleaza anual”
⇒1986 – primul CI pe 222 elemente (de 4 biti)

1968 - firma Intel a luat fiinta (Robert Noyce ← Gordon Moore)
inventator al CI

1969 - primul RAM static – INTEL

1970 - primul RAM dynamic – INTEL

1971 - primul microprocessor – INTEL 4004 pe 45 date, 125 adrese, ceas de 740 KHz, PMOS
- primul EPROM – avantajul posibilitatii stergerii informatiei cu raze ultraviolete TED HOFF - inventator al microprocesorului
- microprocesorul a fost privit ca un circuit programabil ce poate inlocui logica cablata

1972 - microprocesorul 8008 Intel:
- 8 biți date, 14 biți adrese (16 Koct memorie adresabila), ceas 800 KHz
- set de instructiuni simple pentru operatii aritmetice (adunare, scadere) si logice pentru operanzi pe 85
- stiva implementare hardware, mica

1973-1977 - perioada specifica microprocesorului pe 8 biți si adrese 16 biți
- set de instructiuni extins pentru manipulare date pe 16 biți (se pot evalua adrese)
- stiva mutata din CPU in memorie ⇒ posibilitatea de manipulare a subrutinelor imbricate
- limbaj de programare → limbaj de asamblare
- posibilitate de utilizare a perifericelor specializate in aritmetica intregilor extinsa pe axa numerelor reale in format VM → dar acestea aveau viteza mica ⇒ consumuri suplimentare (overhead) mari
- reprezentant tipic microprocesorul 8080 (1974) :
- domenii de aplicatie : sisteme in timp real, sisteme pentru conducerea proceselor industriale
- tehnologii de realizare NMOS ⇒ viteza mare ceas 2- 4 MHz ⇒ viteza mare de executie a instructiunilor
- alte microprocesoare – Intel 8085, Z80 (ZILOG)
- Intel 8086 - 16 biți date, 20 biți adrese, ceas 4 – 8 MHz
- intel 8088 - compatibil cu circuite existente la acea vreme(1979)⇒ idem 8086 dar date pe 8 biți

Caracteristicile microprocesoarelor de la inceputul anilor ‘80
- extinderea lungimii cuvintelor de date la 16 biți si adrese 20 biți (1 Moct adresabil)
- aritmetica numerelor intregi extinsa la 32 biti (usurinta in manipularea adreselor)
- s-au introdus instructiuni pentru manipularea sirurilor cu lungime variabila de biti
- s-a perfectionat mecanismul CALL/ RETURN, cu posibilitati sporite de implementare a limbajului HLL si recursive
- au aparut noi tipuri de coprocesoare pentru prelucrarea in VM
- avans in tehnologia de realizare CMOS

1982 - Intel 286 –16 biți date, 24 biți adrese (16 Moct), ceas 6 ÷ 16 MHz

1984 - apare primul calculator PC AT cu microprocesor 286 la baza
- cea mai importanta facilitate – cea legata de protectia memoriei ceea ce presupune o organizare hardware speciala destinata oferirii de suport pentru taskuri specifice sistemului de operare
- s-au dezvoltat noi tipuri de aplicatii :
  - legate de calculatoarele personale
  - statii de lucru CAD
  - sisteme functionand cu divizarea timpului (time sharing)
- aplicatiile datorita gradului de paralelism oferite in executia instructiunilor se scriu in limbaje de nivel inalt (HLL)
- posibilitatea accesarii de memorie virtuala – implementat deocamdata software ptr MMU

1985 - Intel 386 - 32 biți date
- cu FPU integrat pe cip
- cache – incorporat

1989 - Intel 486 - viteza > 386
- cu FPU integrat pe cip
- cache – incorporat

1993 - Pentium - frecventa 60 – 233 MHz
- 2 benzi de asamblare
- componenta MMX – unele modele mai noi

1995 - Pentium Pro - memorie intermediara pe 2 nivele incorporata
150 – 200 MHz

1997 - Pentium II - Pentium PRO+ MMX
- >233 MHz → Celeron – performanta scazuta
→ Xeon – profesional

1999 - Pentium III - 400 - 700 MHz

2000 - Pentium IV (Itanium)
- frecventa > 1,4 GHz
- arhitectura Netburst (suport pentru Internetul vizual)
- ALU – se numeste Rapid Execution Engine
- se prelucreaza comenzi la jumatate de ciclu de tact
- Pentium IV la 1,4 GHz = 3x mai rapid la Pentium III la 1000 MHz = 7% mai rapid ca Pentium III la 1400 MHz (29 milioane tranzistori)
- 42 milioane tranzistori
- support Internet → schimburi de date direct intre PC- urile utlizatorului Internet
→ strategia Nepster

Caracteristicile ultimelor microprocesoare

- bazate pe existenta a 4unitati majore :
  - CPU (unitatea centrala de prelucrare)
  - FPU (unitatea de prelucrare in VM)
  - MMU (unitatea de gestiune a memoriei)
  - MMX (unitatea multimedia)

CPU - pentru executarea lucrarilor de uz general
FPU - specializata in operatii aritmetice in VM
MMU - suport pentru functii de memorie virtuala si suport hard pentru protectia memoriei
MMX - suport pentru functii multimedia de prelucrare video si sunet

- pe langa acestea, ultimele microprocesoare mai cuprind pe placheta circuite ce inainte erau exterioare : - controller de memorie cache
  - coprocesor de periferice pentru operatii rapide de I/O
  - suport pentru grafica de mare viteza
  - suport pentru multiprocesare

Concluzie :

Aria disponibila a pastilei de siliciu s-a utilizat in 2 scopuri :
1. oferirea de suport pentru implementarea sistemelor complexe de operare prin introducerea de harware specializat si instructiuni specifice
2. executia eficienta a programelor scrise in HLL prin utilizarea unor moduri de adresare si a unor instructiuni mai complexe.

Exemple de microprocesoare CISC – Intel

Facilitati speciale:

286
- gestiunea memoriei virtuale
- implementarea de medii protejate pentru executie
- gestiunea task-urilor in regim de multiprocesare
- gestiunea memoriei - se face de MMU integrata cu implementări speciale pentru cod, date, stiva, date suplimentare
- teste implementate hard - privind corectitudinea accesului la memorie : - limita segmentelor
  - separarea task-urilor de cele privilegiate
  - validitatea indicatorilor si operatiilor cu subrutine

386
- viteza mai mare
- facilitati de optimizare in regim multitasking si suport SO
- spatiu de adresare marit
- posibilitati de testare si depanare

486
- conducta de execuție (pipeline)
- cache pentru date
- coprocesor aritmetic pe chip

Pentium
- integrare inalta (3 milioane tranzistori si trasee $\approx 0{,}5\ \mu m$)

- arhitectura superscalara – prin executarea instructiunilor cu tehnologie pipeline

2 conducte in 5 trepte → prefetch
→ de code1
→ de code2
→ executie
→ writeback

⇒ mai multe instructiuni pe ciclu instructiuni sunt executate

- anumite instructiuni sunt cablate nu microcodate
- cache suportat pentru cod si date de cate 8 Koct. – ascoiative cu 2 cai pentru cautarea datelor pe 32 biti fara a baleia intreaga memorie

cache de date – foloseste 2 tehnici:

→ writeback – ce transfera datele in memoria principala numai cand sunt solicitate prin metoda write though – se transferau de cate ori erau rescrise in cache ⇒ performanta sistemului prin reducerea utilizarii magistralelor

→ protocolul MESII (Modified Exclusive Shared Invalid) – pentru consistenta datelor

- predictia Branch – ce se refera la salturile pentru continuarea executiei unei branch sau iesirea din el si se bazeaza pe predictia ca branch-ul anterior se va folosi din nou. Se folosesc doua buffere de prefetch

→ unul realizeaza prefetch-ul pentru urmatoarea instructiune

→ altul pentru adresa de la inceputul buclei

- FPU incorporat de mare performanta cu 8 trepte de conducta (3 trepte pentru instructiuni in VM sunt adaugate celor 5 trepte ale conductelor intregilor)

- functiile comune + , *, / sunt cablate

- magistrale de date de 64 biti
- e implementat ciclul de magistrala in conducta (bus cycle pipeling) – pentru cresterea latimii de banda a magistralei permite inceperea unui al doilea ciclu cand primul nu s-a terminat ⇒ memoria are mai mult timp pentru decodificarea adresei → se poate folosi cu componente mai lente
- are 2 buffere pentru scriere corespunzator fiecarei conducte
- integritatea datelor prin protejarea si asigurarea integritatii lor. E asigurata de integrarea a doua caracteristici

→ detectia erorilor interne – prin biti de paritate in memoria interna

→ testarea redundantei functionale FRC (Functional Redundancy Check) – prin folosirea a 2 Pentium, unul master, altul checker (verificator), iesirile sunt comparate si diferit anuntate

- suport pentru multiprocesare sporit prin:

→ controller de intreruperi pe chip

→ modul de lucru dual procesor - in care doua procesoare impart memoria cache secundara

- suport de paginare sporit (4 Koct. sau mai mult) pentru scaderea swapping-ului

Arhitectura - unitati componente :
- CPU - IU – Instruction Unit - decodificare instructiuni si stocare in sirul de instructiuni decodificate
  - EU – Execution Unit – 8 registrii de uz general pentru manipulare date, adrese ……
- MMU Memory Manager Unit
  - SU – Submit Unit
  - PU – Page Unit – pagina de 4 Koct. suprapuse peste segmente
- BIU Bios Interface Unit
- FPU Float Point Unit

Registre
- de uz general
- segement
- flag-uri
- IP
- comanda CR∅ ÷ CR3
- adresa interna
- depanare
- testare TR6, TR7

Moduri de lucru
- → real
- → protejat

Tipuri de date
- bit
- camp de biti
- intreruperi pe 16, 32, 64 biti
- offset 16, 32 biti
- indicator : selector + offset
- ASCII
- BCD impartit , neimpartit
- VM

Moduri de adresare
- directa prin registrii G(g)
- cu autoincrementare M(M(PC)) ; PC=PC+1
- imediata M(PC) ; PC=PC+1
- indirecta prin registrii M(G(g))
- indirecta bazata cu deplasare M(G(g)+d)
- indirecta bazata indexata M(G(g1)+ G(g2)*Sb*N) unde N= 0, 1, 2, 4

3.5.2. UltraSPARC II

1970 - Andy Bachtolsheim – la Univerisitatea Stanford a realizat prima statie de lucru in retea numita SUN 1 (Standford University Network) → cu microprocessor Motorola 68020

1982 - fondare Sun Microsystem SUN2, SUN3 – folosesc microprocessor Motorola
- cu placa de retea conexiune Ethernet, soft TCP/ IP

1987 - Sun realizeaza primul UCP propriu cu tehnologii RISC II numit SPARC (Scalable Processor ARChitecture = arhitectura scalabila de procesor)

masina pe 32 biti la 36 MHz
instructiuni pentru intregi

1995 - UltraSPARC – versiunea 9 (V9)
- pe 64 biti
- prelucrare, imagini si semnale audio/video
- set de noi instructiuni numite VIS = Visual Instruction Set
- destinat aplicatiilor profesionale ca servere Web mulltiprocesor cu zeci de UCP-uri si memorie fizica de ordinul 2 TB (1 terabyte = 1012 octeti)

UltraSPARC I –
cu frecventa crescuta de ceas

UltraSPARC II –

3.5.3. Cipuri PicoJava

- Sun defineste o masina virtuala numita JVM cu memorie de 32 biti, 226 instructiuni.
- exista compilator care compileaza ptr JVM.
- exista interpretor pentru masina compilatoare

(compilare + executie)

majoritatea browserelor au interpretor Java pentru a executa appleturi

- exista cipuri JVM hardware – pe care lucreaza direct programe executabile JVM fara a necesita interpretare sau compilare

firma Sun – PicoJava II

4. DESCRIEREA UNITĂȚII CENTRALE DE PRELUCRARE. STUDIU DE CAZ: INTEL 8086

3.3. ARHITECTURA ŞI FUNCȚIONAREA UCP

3.3.1. Componente funcționale

UCP include următoarele unități funcționale:

- **UAL (Unitatea Aritmetica si Logica)** reprezinta o unitate combinațională cu **două intrări şi o ieşire care executa operații aritmetice si logice.**
- **UC (Unitatea de Comanda)** este unitatea functionala care programează execuția secvențială a tuturor operațiilor necesare efectuării instrucțiunilor, generând semnale de comandă pentru tot sistemul, dirijând fluxul de date, corelând viteza de lucru a unității centrale de prelucrare cu memoria, etc.. Activitatea unității de comandă este pilotată de un *semnal de ceas* a cărui frecvența este acum de ordinul sutelor de MHz.
- **Registrii** Aceştia reprezintă elemente de memorare în care se stochează temporar date sau adrese. Unii sunt folosiți pentru urmărirea execuției instrucțiunilor (registrul contor de program), alții sunt folosiți în calcule (regiştri cu scop general), altul păstrează starea programului în execuție (registru de stare), alții pentru calculul adreselor de memorie (regiştri de adresă). Aceştia reprezintă cea mai rapidă formă de memorie din sistem fiind direct conectați la UAL.

Efectuarea transferurilor de date şi comenzi intre unitatile functionale ale microprocesorului se face pe **magistrala internă de date** a microprocesorului.
Semnalele electrice prin care microprocesorul dă comenzi de execuție către memorie şi către celelalte componente din sistem se numesc **semnale de comandă**.
Semnalele electrice prin care microprocesorul culege informatii privind starea componentelor din sistem se numesc **semnale de stare**.

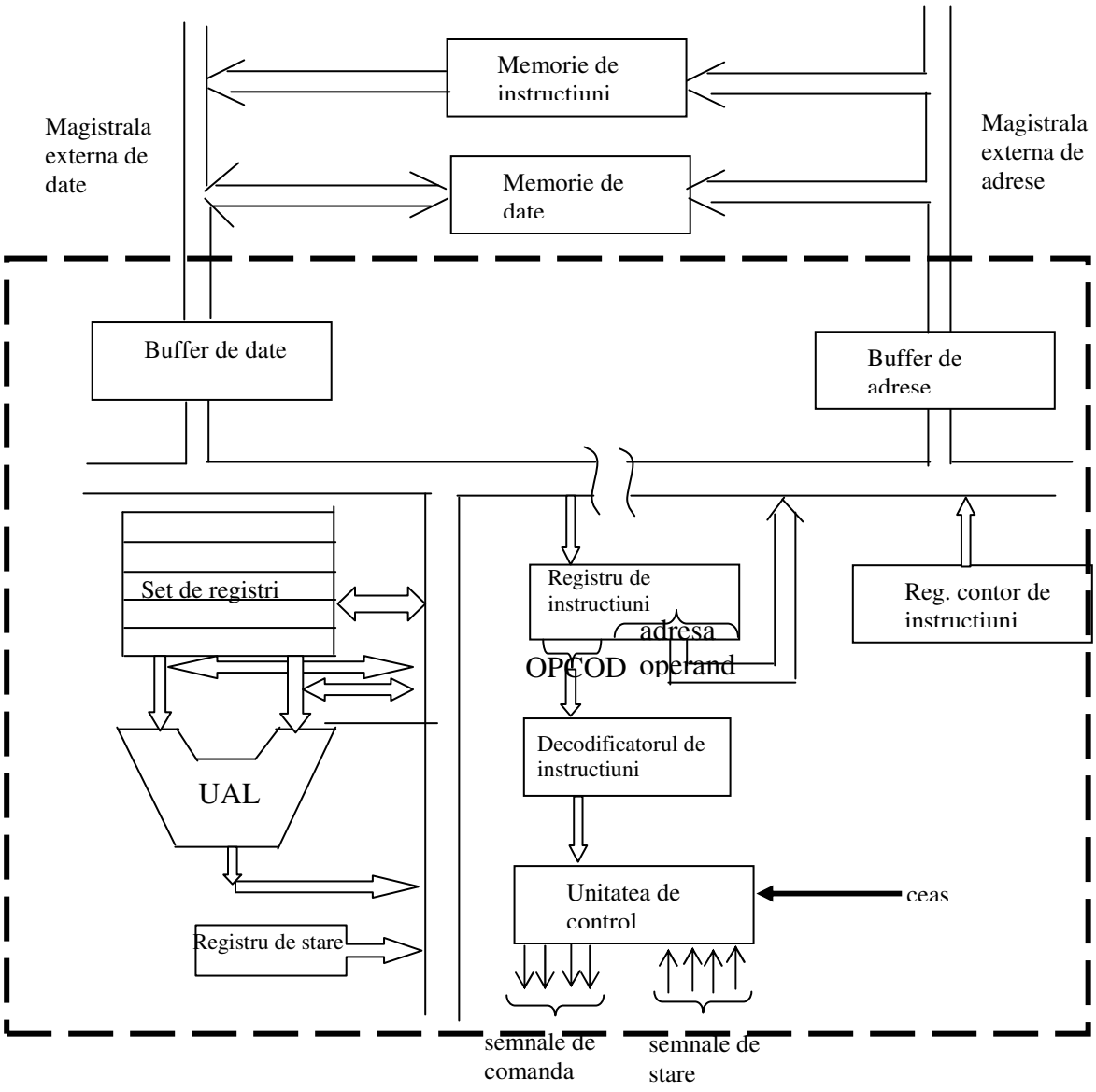
Lungimea (numărul de biți) regiştrilor interni se corelează de obicei cu *lățimea* (numărul de linii) ale magistralei de date. Aceasta e *măsura numărului de biți* ai microprocesorului. Microprocesoarele cu structurv fixă sunt de 8,16,32,64 biți. Lungimea de cuvânt a microcalculatoarelor realizate cu microprocesoare « bit slice » (felii de bit), a căror structură e flexibilă, va fi un multiplu întreg al numărului de biți ai unei felii.
Registrul de adrese, respectiv lățimea magistralei de adrese defineşte spațiul de memorie adresabil direct de microprocesor . O magistrală de adrese de 16 biti permite adresarea a 2^{16}=65536 celule distincte, iar 20 linii de adresă ne duc în lumea megaocteților: 2^{20}=1.048.576 celule adresabile.

Cuvantul instuctiune contine un numar de biti ce exprima cimpul codului operatiei (**OPCODE**) si un cimp corespunzator valorii sau adresei operandului.

3.3.2. Funcționarea UCP

Programul executat de microprocesor se află stocat în memorie în format codificat. Fiecare instrucțiune este codificată corespunzător formatului anterior specificat

(OPCODE şi operand). Instrucțiunile sunt memorate în ordinea de execuție. Un registru special păstrează ordinea de execuție a instrucțiunilor prin adresa urmatoarei instrucțiuni de executat. Acesta este registrul contor de instrucțiuni, numit şi PC (Program counter) sau numărător de adrese. Pentru executarea unui program, PC se încarcă cu adresa primei instrucțiuni de executat. Acestă adresă este trimisă către memorie pentru a se obține codul instrucțiunii de executat. Pentru menținerea liniilor de adresă pe durata citirii memoriei se foloseşte un **registru tampon de adrese AB** (Address Buffer). informația codificată, citită din memorie este depusă într-un **registru tampon de date DB** (Data Buffer). Liniile electrice pe care se generează cuvântul binar de adresă formează **magistrala de adrese**, iar cele dedicate datelor citite/scrise din memorie **magistrală de date**.



Din bufferul de date, instrucțiunea de executat se încarcă în **registrul de instrucțiuni (RI)** şi PC se incrementează/actualizează automat păstrând adresa următoarei instrucțiuni de executat. RI păstrează instrucțiunea pe toată durata ei. Codul operației (OPCODE) se transmite decodificatorului de instrucțiuni pentru identificare şi apoi unității de comandă pentru execuție. Câmpul operand conține valoarea operandului, adresa sau alte informații necesare pentru obținerea valorii acestuia. In funcție de necesități valoarea din acest camp este transmisă în sistem sau unității de comandă.
Când toate informațiile necesare sunt disponibile, unitatea de comandă execută instrucțiunea şi adresa urmatoarei instrucțiuni este transmisă din PC pentru executare.

3.3.3. Descrierea unităților funcționale

UAL este circuitul din structura microprocesorului care proceseaza informatia realizând operatii aritmetice si logice. Este un circuit combinational cu doua intrari si o iesire si necesita registre de memorare temporara atât pentru intrari cât si pentru iesire. Rezultatul operatie este stocat in tot intr-unul din registrii de intrare. Acesti registrii se numesc de tip acumulator.

1. **Numaratorul de adrese** (program counter, PC) pastreaza adresa locatiei care contine urmatoarea instructiune executat. Programul este stocat in memorie ca o succesiune de instructiuni ce trebuie executate secvential de catre microprocesor. La executarea unei instructiuni, continutul PC este automat marit cu o unitate pentru a indica urmatoarea instructiune de executat. Exista posibilitatea de prescriere a registrului PC, i.e. se introducere a altor valori decât celor obtinute prin ordinea naturala (salturi in program necesare in decizii si bucle). Pentru initierea unui program H incarcat PC cu adresa de inceput corespunzatoare .Prin comanda de RESET aplicata MICROPROCESORULUI, numaratorul de programe este incarcat cu o adresa fixata de catre producator, in general aceasta este OOOOH. Comanda RESET (initializare) se da automat la punerea in functiune a microprocesorului dupa conectarea tensiunii sau se aplica din exterior.
2. **Registrul de adresare a memoriei**. Acest registru tampon de adresare denumit buffer de adresare e conectat la magistrala de adresare a memoriei, sau a posturilor de I/O. Continutul registrului PC e transferat in bufferul de iesire care va aplica pe magistrala exterioara de adresare un cuvant binar de un bit ce reprezinta adresa unei locatii de memorie sau adresa unui port de I/E. Dar incarcarea buffului de adresare se poate face nu numai la PC, cât si de la alte elemente ale microprocesorului rezulta ca pe magistrala de adresare se pot aplica si cuvinte de adresa diferite de continutul registrului PC. Unele instructiuni pot incarca registrele de iesire cu o adresa rezultata din continutul lui PC la care se adauga sau se scade un numar rezultat in urma anumitor calcule (adesea generind multiple variante de adresare).
3. **Registrul de I/O** (buffer I/O). Prin acest buffer de I/O se realizeaza legatura dintre magistrala de date interioara a microprocesorului si magistrala de date exterioara a sistemului, deci vehiculeaza curentele de date si instructiuni.
4. **Registrul de instructiuni**, RI. Dupa ce un cuvant instructiune e adus din memorie prin bufferul de I/O pe magistrala interna a microprocesorului. O copie a acestui cuvant va fi inscrisa in registrul de instructiuni. Registrul RI pastreaza instructiunea pe durata executarii acesteia. O data copiata instructiunea in RI continutul numaratorului de adrese este automat incrementat cu o unitate PC+1. Instructiunea este divizata in 2 câmpuri: câmpul codului operatiei, OPCODE si câmpul operandului(sau adresei operandului). Bitii din codul operatiei se aplica

decodificatorului instructiunii care, apoi prin unitatea de control, va genera toate semnalele de control necesare executiei instructiunii reprezentata de codul operatiei. Câmpul adresa operandului se aplica bufferului de adresare pentru a forma adresa din memorie unde se afla operandul necesar operatiei.

5. **Regiştri de tip acumulator** Acesştia sunt regiştri, din structura µp, cu cea mai frecventa utilizare. În acestia se pătrează operanzii expresiilor aritmetice sau logice. Rezultatul operației efectuate de UAL se depune in unul din registrii de intrare, alterand continutul vechi al registrului. Microprocesorul permite efectuarea unor operatii (cu un singur operand) folosind aceşti regiştri: stergerea acumulatorului (toti bitii pusi pe 0), inscrierea tuturor bitilor cu valoarea 1 , deplasarea dreapta , stânga, complementarea continutului etc.
6. **Registrul indicatorilor de conditii.** Prin aceastã denumire se intelege un grup de bistabile (flaguri, fanioane) asamblate sub forma unui registru si citite simultan vor genera împreuna cuvântul de stare al programului PSW (Program Status Words). Bitii cuvântului de stare sunt înscrisi la valoarea 1 in urma unor teste din timpul executiei operatiilor aritmetice si logice ale programului.Setul de instructiuni contine si instructiuni condiționale (instructiunea se executã numai dacã fanionul respectiv e setat). O instrucțiune condițională e utilizată pentru realizarea unei ramificații (salt) in program, adicã se schimbã succesiunea de citire din ordinea naturala a instructiunilor de memorie, prin încarcarea PC cu o anumitã adresã .

Unitatea de control - e partea care supervizeaza functionarea corecta a sistemului de calcul. Comenzile generate de unitatea de control se obtin in urma decodificarii instructiunilor, a cererilor de intrerupere (primite de la elementele microsistemului) si a impulsului de ceas.
Modalitatile de implementare a unitatii de control sunt:

- prin microprogramare. Instructiunile reprezintă microprograme, alcătuite din microinstructiuni (inscrise intr-o memorie speciala) executate prin interpretare.
- Hardware – instructiunile sunt direct executate de harware. Se spune ca unitatea de comanda este cablata.

3.4. ORGANIZAREA REGISTRILOR ŞI A MEMORIEI

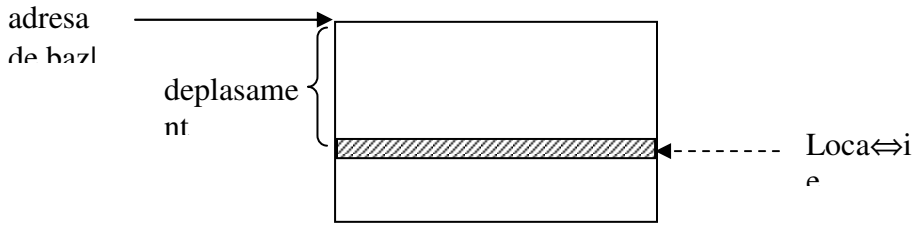
Modul real de funcționare este modul de lucru al microprocesorului I8086.

Adresarea memoriei

Memoria program este împărțită în 4 segmente de memorie corespunzător tipului datelor stocate. Acestea sunt:

- Segmentul de *cod*
- Segmentul de *stivă*
- Segmentul de *date*
- Segmentul de date suplimentar numit *extrasegment*.

Aceste segmente sunt de 64 kocteți fiecare.
Pentru referirea adreselor de memorie se folosesc *regiştri segment* şi *registri offset*.
Registrii segment păstrează adresa de început a segmentului (adresa de bază, AB), iar registrii offset (depl.) deplasamentul în segment.



Dimensiunea registrilor este de 16 biți, iar adresa de memorie de 20 biți.

Adresa efectivă (AE) de memorie se obține astfel:

AE = AB0000+depl.

Unde AB0000 reprezintă adresa de bază deplasată la stănga cu patru de zero. Se obține astfel o adresă pe 20 de biți.

Referirile în segmentul curent, numite de tip *near*, se fac specificându-se doar deplasamentul în acelaşi segment, iar referirile la segmente exterioare, de tip far, se fac specificându-se şi adresa de bază şi deplasamentul în segmentul respectiv.

Registrii segment sunt:

CS (Code Segment) – registrul segment de cod
SS (Stack Segment – registrul segment de stivă
DS (Data Segment) – registrul segment de date
ES (Extra Segment) – registrul segment extra de date

Registrii offset asociați sunt:

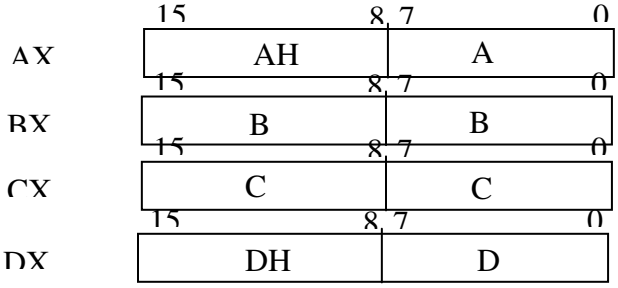
IP (Instruction Pointer) – contorul de program sau indicatorul de instrucțiuni
SP (Stack Pointer) – indicatorul de vârf al stivei
BP (Base Pointer) – indicator de bază (folosit în modul bazat de adresare)
SI (Source Index) – indexul sursă
DI (Destination Index) – indexul destinație.

Deci pentru a referi o valoare din:
- segementul de cod se vor folosi CS şi IP

- segementul de stivă se vor folosi SS şi SP sau SS şi BP
- segementul de date se vor folosi DS şi SI
- segementul de date extra se vor folosi DS şi DI

Regiştri generali

Regiştrii cu scop general participă la operații aritmetice sau logice. În ei se stochează



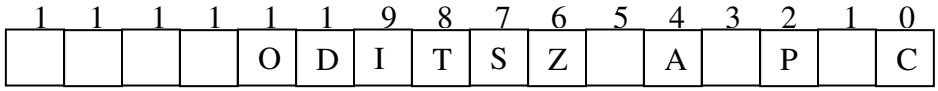
operanzii şi rezultatele oparațiilor. Sunt patru regiştri pe 16 biți: AX, BX, CX, DX.
Se poate lucra cu regiştri de un octet AH, AL, BH, BL, CH, CL, DH, DL.

Microprocesoarele Intel >I386 au regiştri pe 32 biți : EAX, EBX, ECX, EDX în care cuvântul low (cei mai puțin semnificativi 2 octeți) sunt regiştri AX, BX, CX, DX.

Unii din regiştri generali sunt folosiți implicit în unele operații. Astfel:
- registrul AX este registrul acumulator (folosit pentru a păstra rezultatul operațiilor aritmetice sau logice şi pentru operații de transfer date la/de la porturile de I/E.
- registrul BX este folosit ca registru de bază în modul de adresare a memorie bazat
- registrul CX este folosit ca contor
- registrul DX este folosit în operațiile de înmulțire şi împărțire şi pentru adresarea porturilor de I/E.

Registrul de stare sau registrul indicatorilor de condiții

Poziția indicatorilor de condiții in registrul de stare este următoarea:



Indicatorii de condiții sunt următorii:
C - indicatorul de transport (carry)
= 1 indica existența unui transport dela/la cel mai semnificativ bit al rezultatului
= 0 indica ca nu exista un transport dela/la cel mai semnificativ bit al rezultatului
P - indicatorul de paritate (parity)
= 1 indica existența unui număr par de biți de acelaşi fel ai rezultatului
= 0 indica existența unui număr impar de biți de acelaşi fel ai rezultatului
A - indicatorul de transport auxiliar (auxiliary carry)
= 1 indica existența unui transport dela/la digitul (nibble) mai puțin semnificativ la cel mai semnificativ al rezultatului
= 0 indica ca nu exista un transport dela/la digitul (nibble) mai puțin semnificativ la cel mai semnificativ al rezultatului
Z - indicatorul de zero (zero)
= 1 indica o valoare zero ca rezultat
= 0 indica un rezultat diferit de zero
S - indicatorul de semn (sign)

= 1 indica un rezultat negativ
= 0 indica un rezultat pozitiv

O - indicatorul de depăşire aritmetică (overflow)

= 1 indica existența unei depăşiri aritmetice (marimea rezultatului depăşeşte capacitatea de stocare a registrului rezultat)

= 0 indica că nu existentă o depăşire aritmetice

Aceşti indicatori se poziționează după operațiile aritmetice şi logice, corespunzător rezultatului.

Indicatorii de condiții D, I, T se mai numesc şi de control pentru sunt setați prin program.

D - indicator de direcție (direction)

= setat pe 1 produce autodecrementarea în operațiile cu şiruri

= setat pe 0 produce autoincrementarea în operațiile cu şiruri

I = indicator de întrerupere (interrupt)

= setat pe 1 activează sistemul de întreruperi

= setat pe 0 dezactivează sistemul de întreruperi

T = indicator de trap (trap)

= setat pe 1 pune procesorul în mod single step (microprocesorul se opreşte după execuția fiecărei instrucțiuni).

3.5. Ciclul instrucțiune

Executia instructiunilor se realizeaza intr-o secventa de pasi numita **ciclu instructiune**.

Un ciclu instructiune este compus din mai multe cicluri masina. Ciclul esential in functionarea oricarei instructiuni este ciclul extrage-decodifica-executa (fetch-decode-execute).

Secventa de pasi este urmatoarea:

1. Se transfera instructiunea urmatoare (indicata de PC) in registrul de instructiuni.
2. Se schimba PC a.i. sa contina adresa urmatoarei instructiuni de executat.
3. Se determina timpul instructiunii extrase.
4. Daca in instructiune se foloseste continutul unei locatii de memorie, se gaseste.
5. Se executa instructiunea

- prin interpretare - microprogram – care este continut in memorii rapide numai pentru citire numite memorii de control in care se stocheaza microinstructiunile) sau
- direct de catre hardware (cablat).

6. Se reia pasul 1.

Microprocesorul mai executa si alte cicluri masina:

- ciclul de citire din memorie
- ciclul de scriere in memorie
- ciclul de calcul.

3.6. Caracteristici arhitecturale.

Caracteristicile ahitecturale cu cele mai puternice influențe asupra eficienței de execuție a programelor scrise în limbaje de nivel înalt sunt*:*

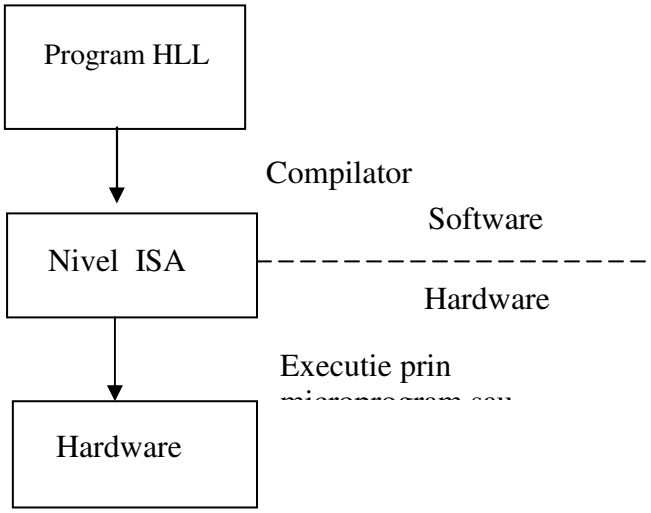
- *tipurile de date* – deoarece ele pot fi aceleaşi su cele ale maşinii
- *modurile de adresare* – ele definesc mecanismul de access la date şi pot fi utilizate eficient pentru reprezentarea structurilor complexe de date
- *setul de instrucțiuni* – reflectă operațiile tipice cerute de execuția programului

Alte caracteristici arhitecturale ale microprocesoarelor sunt:

- *organizarea registrelor*
- *aritmetica numerelor în virgulă mobilă*
- *întreruperi şi capcane*
- *mijloace de depanare.*

3.7. Nivelul arhitecturii setului de instructiuni = ISA (Instruction Set Architecture)

Acest nivel se află între nivelul microarhitecturii şi cel al sistemului de operare.
El reprezintă interfața dintre hardware şi software.



Nivelul ISA al calculatoarelor moderne este compatibil cu cu variantele anteriaoare (**backward compatible**).

Nivelul ISA este definit ca modul în care maşina este văzută de un programator în limbaj de asamblare. Acesti progamatori trebuie să cunoască modelul de memorie, setul de registre, tipurile de date şi instrucțiunile disponibile.

Toate calculatoarele împart memoria în celule cu adrese consecutive. Dimensiunea locației de memorie este de 8 biți (octet). Octeții sunt grupați în cuvinte de 2, 4, 8 oct. Reprezentarea acestora în memorie se face în format **little endian** (calculatoarele cu microprocesoare Intel) sau **big endian** (calculatoarele RISC).
Abilitatea de a citi cuvinte de la adrese arbitrare necesită o logică suplimentara în cadrul cipului (MMU).
Microprocesoarele Intel actuale au trei moduri de lucru:

- Modul real – cu toate facilitățile legate de 8086
- Modul virtual 8086 – in care face posibilă execuția programelor vechi 8088 în mod protejat. SO controlează întreaga maşină.
- Modul protejat – modul de lucru al microprocesorului performanțe crescute oferite de construția acestuia. Aici intră instrucțiunile complexe, mai ales cele MMX (multimedia).

3.8. Setul de instrucțiuni al microprocesorului I8086

Setul de instrucțiuni reprezintă o caracteristică foarte importantă a microprocesorului.

La microprocesorul I8086, setul de instrucțiuni este organizat în 6 grupe.
1. instrucțiuni pentru transferuri de date
2. instrucțiuni aritmetice
3. instrucțiuni logice
4. instrucțiuni pentru manipularea şirurilor de caractere
5. instrucțiuni pentru controlul transferului programului
6. instrucțiuni pentru controlul procesorului.

1. Instrucțiuni pentru transferuri de date

Se împart în patru clase:

a) cu scop general
b) specifice cu acumulatorul
c) cu obiect adresă
d) **referitoare la indicatorii de condiție**

Mnemonica generală a instrucțiunilor este

nume_instrucțiune destinație, sursă

Operanzii destinație pot fi:
- regiştri ai microprocesorului (*reg*)
- locații de memorie (m*em*)

Operanzii sursă pot fi:
- regiştri ai microprocesorului (*reg*)
- locații de memorie (*mem*)
- date constante (data)

1.a. Instrucțiunile pentru transferuri de date cu scop general sunt:
- de tip **MOV** - transferă o valoare din sursă în destinație (realizează o copie a sursei în destinație) - reprezintă instrucțiunea de **atribuire** în limbaj de asamblare

  - sintaxa: *MOV dest,sursa*
  - instrucțiune are multe limitări =>tipuri de instrucțiuni de tip MOV:

```
mov    reg, reg
mov    mem, reg
mov    reg, mem
mov    mem, immediate data
mov    reg, immediate data
mov    ax/al, mem
mov    mem, ax/al
mov    segreg, mem16
mov    segreg, reg16
mov    mem16, segreg
mov    reg16, segreg
```

Observații referitoare la instrucțiunea MOV:
- nu afectează indicatorii de condiție

unul din operanzi se află întotdeauna într-un registru.

- nu sunt admise transferuri din memorie în memorie
- nu se pot încărca regiştrii segment cu valori imediate de date
- unele instrucțiuni MOV sunt mai rapide decât altele (de ex. MOV ax,mem e mai rapidă decât MOV reg, mem)
- operanzii trebuie să fie de acelaşi tip şi pot fi de dimensiunea octeților, cuvintelor sau dublu cuvintelor (>I386). Se poate specifica dimensiunea transferului în instrucțiunea

  Mov dacă se foloseşte sintaxa:

  ```
  mov     byte ptr [bx], 0
  mov     word ptr [bx], 0
  mov     dword ptr [bx], 0 – numai >I386
  ```
- când încărca în regiştri valori mai mici decât dimensiunea regiştrilor, partea mai puțin semnificativă se stochează în partea low a registrului
- pentru încărcarea constantelor în regiştri segment se pot folosi de exemplu instrucțiunile:

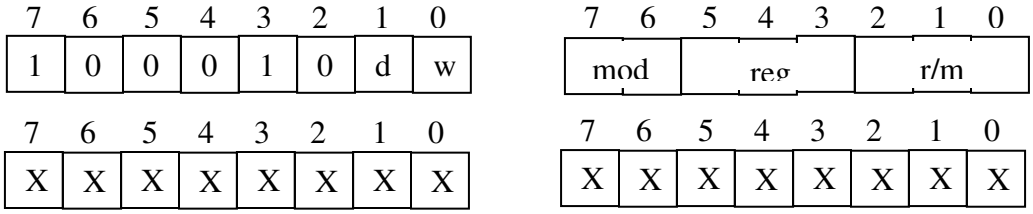
  ```
  mov     ax, 40h
  mov     es, ax
  ```

Pentru setarea flagurilor de control se pot realiza următoarele operații:

- salva starea regiştrilor în stivă
- scoate din stivă într-un alt registru
- modifica valoarea flagului din registrul respectiv
- salva registrul în stivă
- scoate din stivă în registrul de stare

CODIFICAREA INSTRUCȚIUNILOR

Codificarea instrucțiunii MOV este poate cea mai complexă din tot setul de instrucțiuni



unde d=direcție - specifică dacă data este stocată în memorie sau în registru

r/m - reprezintă modul de adresare

w - reprezintă lățimea registrului (1 octet sau 2 octeți)

reg - specifică registrul

mod - codifică direcția transferului (reg->mem sau mem->reg) şi dimensiunea deplasamentului (de 0, 1 2 sau 4 octeți)

- instrucțiuni cu stiva (PUSH, POP)

instrucțiunile push and pop instructions manipulează date cu stiva. Există 19 varietăți de instrucțiuni push:

```
push    reg16
```

```
                pop     reg16
        push    reg32           (3)
        pop     reg32           (3)
        push    segreg
        pop     segreg          (except CS)
        push    memory
        pop     memory
        push    immediate_data  (2)
        pusha                   (2)
        popa                    (2)
        pushad                  (3)
        popad                   (3)
        pushf
        popf
        pushfd                  (3)
        popfd                   (3)
        enter   imm, imm        (2)
        leave                   (2)
```

(2)>I286
(3)>I386

- **instrucțiuni XCHG**

Instrucțiunea xchg (exchange) instruction interschimbă două valori.. The general form is

```
        xchg    operand1, operand2
```

Sunt patru forme pentru instrucțiunea xchg:

```
                xchg    reg, mem
        xchg    reg, reg
        xchg    ax, reg16
        xchg    eax, reg32          (>I386)
```

Instrucțiunea xchg nu modifică flagurile. Operanzii trebuie să fie de aceeaşi dimensiune. Ca şi la instrucțiunea MOV, operațiile cu registrul ax sunt mai rapide decât cu celelalți regiştri.

1.b) Instrucțiuni specifice cu acumulatorul

În această clasă intră instrucțiunile de transfer de la/la porturile de intrare/ieşire.
Un operand se află întotdeauna în registrul acumulator, iar celălalt reprezintă portul.
Dacă portul este pe 8 biți se foloseşte registrul AL, iar dacă portul este pe 16 biți se foloseşte registrul AX.

Instrucțiunea **in ac, port** citeşte un caracter în registrul acumulator de la portul specificat.
Dacă adresa portului este între 0 şi 255 se specifică direct în instrucțiune. Dacă adresa >255, se încarcă regsitrul DX cu adresa respectivă, iar instrucțiunea are sintaxa:

in al, dx (am presupus un port de 8 biți)

Instrucțiunea **out port, ac** scrie un caracter din registrul acumulator în portul specificat.

O altă instrucțiune din această clasă este instrucțiunea **xlat**. Acestă instrucțiune realizeză o translație folosind registrul **bx** ca pointer al adresei de bază a unui tabel de 256 octeți, **al** ca index în acest tabel, iar valoarea pointată din tabel este întoarsă în registrul registrul **al**.

1.c) Instrucțiuni cu obiect adresă.
In acestă categorie intră instrucțiunile de încărcare a regiştrilor segment. Respectiv

lds reg, mem32 – încarcă registrul **reg** cu cuvântul de la dresa de memorie **mem32**, iar registrul segment DS cu valoarea de la locația **mem32+2**

les reg, mem32 - încarcă registrul segment ES

Aceste instrucțiuni sunt folosite pentru setarea pointerilor de tip far.
O altă instrucțiune din acestă clasă este **lea reg, mem** care încarcă în registrul specificat adresa efectivă de memorie **mem**.

1.d) Transferuri de flaguri
În acestă categorie intră instrucțiunile:

lahf – încarcă registrul ah cu conținutul octetului mai puțin semnificativ al registrului de stare;

sahf - încarcă octetul mai puțin semnificativ al registrului de stare cu conținutul registrului ah;

pushf - salveză în stivă registrul de stare

popf - încarcă registrul de stare cu valoarea din vârful stivei.

2. Instrucțiuni aritmetice

Instrucțiunile aritmetice lucrează cu patru tipuri de date:
- binare fară semn (numere reprezentate, în memorie, binar în mărime);
- binare cu semn (numere reprezentate, în memorie, în mărime şi semn în complement față de doi);
- BCD neîmpachetate fară semn (un număr zecimal reprezentat pe un octet);
- BCD împachetate cu semn (două număre zecimal reprezentate pe un octet);

Când numerele sunt cu semn, bitul de semn este cel mai semnificativ bit al numărului (1 – pentru numere negative, 0 pentru numere pozitive).
Indicatorii de condiție se poziționează corespunzător rezultatului.

Instrucțiunile aritmetice se împart în:
- instrucțiuni pentru adunare
- instrucțiuni pentru scădere
- instrucțiuni pentru înmulțire
- instrucțiuni pentru împărțire

Instrucțiunile aritmetice afectează indicatorii de condiție din registrul de stare.

a) Instrucțiuni pentru adunare:

add dest, sursă - realizeză operația dest+sursa cu depunere în dest

adc dest, sursă - realizeză operația dest+sursa+CF cu depunere în dest, unde CF=carry flag

inc dest - realizeză operația dest+1 cu depunere în dest

aaa – (ASCII Adjust for Addition) realizeză o corecție ASCII la adunare a rezultatului stocat în registrul AL după o operație de adunare ASCII

daa - (Decimal Adjust for Addition) realizeză o corecție BCD la adunare a rezultatului stocat în registrul AL după o operație de adunare BCD

b) Instrucțiuni pentru scădere:

sub dest, sursă - realizează operația dest-sursa cu depunere în dest
subb dest, sursă - realizează operația dest-sursa-CF cu depunere în dest, unde CF=carry flag
dec dest - realizează operația dest-1 cu depunere în dest
neg dest - realizează operația 0-dest cu depunere în dest
cmp dest, sursă - compară operanzii dest şi sursă prin scădere (dest-sursă), poziționează corespunzător indicatorii de condiție din registrul de stare fără să modifice destinația.
aas – (ASCII Adjust for Substraction) realizează o corecție ASCII la scădere a rezultatului stocat în registrul AL după o operație de scădere ASCII
das - (Decimal Adjust for Substraction) realizează o corecție BCD la scădere a rezultatului stocat în registrul AL după o operație de scădere BCD.

c) Instrucțiuni pentru înmulțire

In general un operand se află în registrul acumulator (AX sau AL), iar rezultatul se încarcă tot în registrul acumulator (AX) şi în registrul DX partea mai semnificativă dacă operanzii sunt de doi octeți.
mul dest, sursă - realizează operația dest*sursa fără semn cu depunere în dest. Dest este AL sau AX
imul dest, sursă - realizează operația dest*sursa cu semn cu depunere în dest. Dest este AL sau AX
aam – (ASCII Adjust for Multiplication) realizează o corecție ASCII la înmulțire a rezultatului stocat în registrul A după o operație de înmulțire ASCII

d) Instrucțiuni pentru împărțire

Deîmpărțitul se află se află în registrul acumulator AX sau în AX şi DX iar rezultatul se încarcă în registrul acumulator (AL, AX). Restul împărțirii se întoarce în AH sau DX
div dest, sursă - realizează operația dest/sursa fără semn cu depunere în dest.
idiv dest, sursă - realizează operația dest/sursa cu semn cu depunere în dest.
aad – (ASCII Adjust for Division) realizează o corecție ASCII la împărțire a rezultatului stocat în registrul A după o operație de împărțire ASCII
cbw – (Convert Byte to Word) – permite o extensie de semn a lui AL în AH
cwd – (Convert Word to DoubleWord) – permite o extensie de semn a lui AX în DX

3. Instrucțiuni logice

Se clasifică în:
a) instrucțiuni cu un operand (monadice)
b) instrucțiuni cu doi operanzi (diadice)

a) **instrucțiuni cu un operand** (monadice)
not dest - realizează complementul față de 1 al operandului dest cu depunere în dest.

deplasări:
shl dest, contor - realizează deplasarea logică la stânga a destinației cu un număr de biți specificați de contor
shr dest, contor - realizează deplasarea logică la dreapta a destinației cu un număr de biți specificați de contor
sal dest, contor - realizează deplasarea aritmetică la stânga a destinației cu un număr de biți specificați de contor
sar dest, contor - realizează deplasarea aritmetică la dreapta a destinației cu un număr de biți specificați de contor

rotiri:
rol dest, contor - realizează rotirea logică la stănga a destinației cu un număr de biți specificați de contor
ror dest, contor - realizează rotirea logică la dreapta a destinației cu un număr de biți specificați de contor
rcl dest, contor - realizează rotirea cu carry la stănga a destinației cu un număr de biți specificați de contor
rcr dest, contor - realizează rotirea cu carry la dreapta a destinației cu un număr de biți specificați de contor

b) instrucțiuni cu doi operanzi (diadice)
and dest, sursă - realizează operația şi logic între destinație şi sursa cu depunere în dest
or dest, sursă - realizează operația sau logic între destinație şi sursa cu depunere în dest
test dest, sursă - realizează operația şi logic între destinație şi sursa fără depunere în dest
xor dest, sursă - realizează operația sau exclusiv între destinație şi sursa cu depunere în dest

4. Instrucțiuni pentru manipularea şirurilor de caractere

Pentru toate instrucțiunile cu şiruri de caractere se consideră că şirul sursă este conținut în segmentul curent de date, pointat de regiştri DS:SI, iar şirul destinație în extrasegment, pontat de ES:DI. Pentru sursă se poate considera şi un alt registru segment dacă se foloseşte un prefix de registru adecvat.

Sensul de parcurgere în memorie al şirurilor este indicat de indicatorul de condiție DF (direction flag) din registrul de stare (0 = sens crescător, de la drese mici spre adrese mari, iar 1 = sens descrescător, de la drese mari spre adrese mici).

Un prefix de un octet poate precede instrucțiunile cu şiruri pentru a indica că operația trebuie repetată până sunt îndeplinite condițiile indicate. Exemple:
REP – repetare până CX=0
REPZ – repetare cât ZF=1, până ZF=0
REPNC – repetare cât CF=0, până CF=1

Contorul repetiției se află în registrul CX. Repetiția se termină când CX ajunge 0. Acest test este prioritar testului corespunzător prefixului de repetiție.

În timpul execuției repetiției operației primitive, regiştri SI, DI, CX sunt actualizați după fiecare operație (SI, DI se incrementează sau decrementează pentru a indica următorul element al şirului, iar CX se decrementează). Instrucțiunile pot fi astfel întrerupte şi reluate din punctul respectiv.

Instrucțiunile primitive cu şiruri de caractere sunt:
MOVB ăMOVW – transfera un operand de un octet (cuvânt) de la şirul sursă în destinație
CMPB ăCMPW – compară un operanzii corespunzători din şirul sursă şi destinație, poziționând corespunzător indicatorii de condiție din registrul de stare.

SCAB ăSCAW – compară elementul curent al şirului destinație cu valoarea din registrul AL. Dacă se execută în mod repetat scaneză şirul destinație în căutarea valorii din AL.
LODB ăLODW – transfera un operand de un octet (cuvânt) de la şirul sursă în registrul AL (AX). Acestă operație nu poate fi repetată.
STOB ăSTOW – transfera un operand de un octet (cuvânt) din AL (AX) în şirul destinație. Repetitiv se poate folosi pentru umplerea unui buffer cu o valoare dată.
INB ăINW - citeşte data de la un port de intrare
OUTB ăOUTW – scrie data la un port de ieşire

5. Instrucțiuni pentru controlul fluxului de executie a programului

Se pot imparti in patru clase:

a) trasferuri neconditionale
b) trasferuri conditionale
c) controlul iteratiilor
d) intreruperi software

a) transferuri neconditionate:

CALL eticheta

Instructiune pentru apeluri de subprograme. Se executa subprogramul de la eticheta specificata.

Pasii parcursi in executia subprogramului sunt:

- salvare in stiva a adresei de revenire in programul apelant
- trasfer control (incarcare registru IP cu adresa de inceput a subprogramului).

Exista doua tipuri de apel:

- NEAR – subprogram in acelasi segment de cod cu programul apelant (se indica doar adresa offset)
- FAR - subprogram in alt segment de cod decat programul apelant (se indica adresa segment si adresa offset)

Alte forme pentru CALL:

CALL reg -> adresa de salt se afla in registru
CALL [reg] -> adresa de salt se afla la adresa indicata in registru.

JMP eticheta -Transfer neconditionat la adresa indicata de eticheta.
JMP SHORT nr. – forma scurta a instructiunii JMP care realizeaza un transfer neconditionat la instructiunea aflata cu +/- nr. octeti dupa/inainte de instructiunea curenta.

RET – Instructiune de intoarcere din subprogram care transfera controlul programului la adresa din varful stivei. De obicei aceasta reprezinta adresa de intoarcere in programul apelant.
RET nr. – instructiune de intoercere cu extragerea a nr. parametri din stiva.

b) trasferuri conditionate

Instructiuni care implementeaza structurile alternative ale programarii structurate. Ramificatiile se fac in functie de starea indicatorilor de conditie, iar salturile nu pot depasi +/- 128 octeti de la adresa instructiunii de salt.

JZ eticheta - inseamna Jump on zero si transfera controlul programului la eticheta specificata daca flagul de ZERO =1

JNZ eticheta - inseamna Jump on not zero si transfera controlul programului la eticheta specificata daca flagul de ZERO =0

JC eticheta - inseamna Jump on carry si transfera controlul programului la eticheta specificata daca flagul de CARRY =1

JNC eticheta - inseamna Jump not carry si transfera controlul programului la eticheta specificata daca flagul de CARRY =0

JP eticheta - inseamna Jump on parity si transfera controlul programului la eticheta specificata daca flagul de PARITY =1

JP eticheta - inseamna Jump not parity si transfera controlul programului la eticheta specificata daca flagul de PARITY =0

Sunt implementate instructiuni de acest fel pentru toti indicatorii de conditie.

Alte instructiuni:

JE eticheta – inseamna Jump on Equal si este idem JZ.

JNE eticheta – inseamna Jump not Equal si este idem JNZ.

JL eticheta – inseamna Jump on Less si inseamna salt daca dest<sursa

JG eticheta – inseamna Jump on Greater si inseamna salt daca dest>sursa

JLE eticheta – inseamna Jump on Less or Equal si inseamna salt daca dest<=sursa

…

JCXZ eticheta – inseamna salt la eticheta daca continutul registrului CX este 0.

c) controlul iteratiilor este realizat cu instructiuni de tip LOOP care implementeaza structuri repetitive cu testul la sfarsit.

Adresa de repetitie trebuie sa se incadreze in domeniul +/- 128 octeti de la instructiunea curenta.

LOOP eticheta – decrementeaza pe CX cu 1 si face salt la instructiunea specificata de eticheta daca CX<>0.

LOOPZ eticheta – decrementeaza pe CX cu 1 si face salt la instructiunea specificata de eticheta daca CX<>0 si ZF=1

LOOPNZ eticheta – decrementeaza pe CX cu 1 si face salt la instructiunea specificata de eticheta daca CX<>0 si ZF=0

Se pot utiliza si ceilalti indicatori de conditie pentru instructiunea LOOP cu sintaxa corespunzatoare.

d) **Intreruperi software** – instructiunile genereaza apelul rutinei (subprogramului) de intrerupere cu numarul specificat.

INT nr

Fiecare rutina de intrerupere are

- un numar asociat care refera nivelul intreruperii
- o adresa care la care se afla stocata.

La I8086 exista 256 de nivele de intrerupere.

In memorie exista un tabel asociat intreruperilor numit “tabel al vectorilor de intrerupere”. El contine pentru fiecare nivel de intrerupere patru octeti ce reprezinta adresa de memorie unde se afla stocata rutina respectiva de intrerupere (2 octeti pentru adresa segment si 2 octeti pentru adresa offset).

Primele nivele de intrerupere sunt asociate intreruperilor hardware, intreruperi generate de dispozitivele de I/E din sistem.

Pentru executia instructiunii INT nr (sau a rutinei asociate unei intreruperi hardware):

- se salveaza in stiva registrul de stare si adresa de intoarcere in program
- se calculeaza adresa din tabel corespunzatoare nivelului: nr*4+adresa de baza in tabel si se preda controlul rutinei aflate la adresa respectiva.

Rutinele de intrerupere se incheie cu instructiunea IRET.

IRET – instructiune care incarca registrul de stare si registrii CS si IP cu valorile din varful stivei.

INTO – instructiune care genereaza o intrerupere de nivel 4 daca OF=1.

Instrucțiuni pentru controlul procesorului.

Doua clase:

1. Instructiuni cu indicatorii de conditie

CLC – pune flagul carry pe 0
STC – pune flagul carry pe 1
CMC – complementeaza carry flag
CLD – pune flagul direction pe 0
STC – pune flagul direction pe 1
CLI – pune flagul interrupt pe 0 = dezactiveaza sistemul de intreruperi
STI – pune flagul interrupt pe 1 = activeaza suistemul de intreruperi.

2. Instructiuni referitoare la starea procesorului

HLT- trece microprocesorul in stare de oprire (HALT). Iesirea din aceasta stare se face prin RESET sau activarea unei intreruperi externe.
WAIT – trece microprocesorul in stare de asteptare (WAIT) necesara sincronizarii cu hardware-ul extern.
ESC – utilizata pentru comunicarea cu alte procesoare
BUS LOCK – specifica un prefix de un octet care poate precede orice instructiune si care determina procesorul sa genereze un semnal de blocare magistrala pe durata instructiunii.
Se utilizeaza in operatii multiprocesor.

4. MODURI DE ADRESARE

4.1. Considerații generale

4.1.1. Adrese virtuale

Adresele incluse în programele executate de microprocesoarele moderne (după I8086) nu sunt fizice. Ele sunt relative la spațiul de memorie alocat programului la momentul execuției de către sistemul de operare. Programul conține, deci adrese virtuale care la momentul execuției sunt transformate în adrese fizice. Spațiul de memorie al programului este un spațiu virtual (**memorie virtuală**) necesar pentru execuția programului şi asigurat de sistemul de operare prin infrastructura pusă la dispoziție de microprocesor.

4.1.2. Adresa efectivă la I8086

In plus la I8086 dimensiunea magistralei de adrese este de 20 biți, iar dimensiunea registrelor de adresare este de 16 biți. Pentru a se ajunge la adresa fizică (efectivă) de memorie se realizează următoarele operații:

- adresa de segment (AS) care pointează începutul segmentului de memorie este completată cu patru biți de zero (prin deplasare la stânga), obținându-se o adresă pe 20 de biți
- la acestă adresă se adună adresa offset (deplasamentul din segmentul de memorie)

Adresa efectivă (pe 20 de biți) va fi deci:

AE = AS0000 + AO

4.1.3. Raționamente de reprezentare

În toate limbajele de programare procedurale, datele prelucrate se păstrează în memorie.

Operanzii utilizați în instrucțiunile microprocesorului sunt specificați implicit sau explicit:

- **implicit** de instrucțiune în cazul instrucțiunilor care se referă la un anumit registru al microprocesorului
- **explicit** în instrucțiune - cazul în care în instrucțiune apare operandul.

Modurile de adresare specifica modalitatea explicită de obținere a operanzilor.

Codul instrucțiunii cuprinde două câmpuri:

- câmpul CODOP – ce specfifică operația de executat
- câmpul *operand* – ce specifică în mod explicit operandul.

Câmpul operand trebuie să indice operanzii care se prelucrează (unul sau doi în funcție de operație) şi locul unde se depune rezultatul operației. În cazul extrem câmpul operand ar trebui să codifice doi operanzi şi adresa rezultatului. Pentru a reduce dimensiunea instrucțiunii, la unele microprocesoare, inclusiv Intel, rezultatul este depus la locația unui operand. Astfel se codifică maxim două

obiecte şi de aici instrucțiuni cu dimensiune mai mică şi deci programe care ocupă mai puțin spațiu în memorie.
Operanzii se pot păstra în registre sau în memorie.
Registre constituie modalitatea optimă de păstrare deoarece se codifică pe mai puțin biți şi deci ocupă spațiu redus pentru reprezentare şi operațiile realizate sunt mai rapide fiind direct legați la unitățile de execuție. Din păcate numărul registrelor este limitat. La microprocesorul studiat, numărul de registre este de 8 registre de 8 biți: AL, AH, BL, BH, CL, CH, DL, DH. Cei 8 registre se codifică pe 3 biți astfel:
000 – AL
001 – AH
010
011
100
101
110
111

4.1.4. Structura generală a modurilor de adresare

Modurile de adresare specifică modalitatea de obținere a operanzilor. Păstrarea operanzilor în memorie oferă flexibilitate programatorului şi posibiltăți multiple de obținere a lor. Se poate astfel ca un operand să se găsească la o locație de memorie sau la acea locație să se afle adresa operandului sau adresa adresei operandului. În plus la nivelul microprocesorului se oferă suport pentru implementarea structurilor complexe de date utilizate în limbajele de nivel înalt.
Elementele de bază care se utilizează pentru obținerea oricărui mod de adresare sunt:

- obiecte
- funcții de bază.

Obiectele din modurile de adresare sunt:

- *registre ale microprocesorului*
- *deplasamente* (offset) în segmentele de memorie, specificate în câmpul operand al instrucțiunilor

Registrele microprocesorului au următoarele funcții:

- registru operand – în registru se află valoarea operandului
- registru indirect – în registru se află adresa operandului
- registru de bază – în registru se află o adresă de bază la care se adună offsetul pentru a se obține adresa operandului.

Deplasamentul din câmpul instrucțiunii poate reprezenta:

- valoarea operandului
- adresa operandului
- valoarea deplasamentului care se adună la adresa de bază pentru obținerea adresei operandului.

Funcțiile de bază realizate în modurile de adresare sunt:

- *adunarea* – specifică adunarea mai multor valori pentru a obține adresa operandului
- *indirectarea* – adresa utilizată specifică adresa operandului
- *deplasarea* – utilizată pentru înmulțire cu 2 (2^1), 4 (2^2), 8 (2^3), etc (ce specifică numărul de octeți ai elementelor structurilor de date stocate în memorie) şi utilizată pentru referirea elementelor succesive din memorie.

4.1.5. Problematica modurilor de adresare

Aducerea operanzilor din memorie necesită cicluri suplimentare de citire a memoriei. Un ciclu pentru a obține un operand a cărui adresă se cunoşte, două cicluri dacă se cunoaşte adresa adresei operandului ş.a. Moduri de adresare sofisticate duc la timpi suplimentari de execuție. Avantajul acestor moduri de adresare îl reprezintă creşterea în flexibilitate a programului. Astfel dacă modul de adresare este primar şi valoarea operandului se specifică în program, la schimbarea operandului trebuie schimbat şi programul, recompilat şi linkeditat. Dacă se cunoşte adresa atunci se poate schimba conținutul de la adresa respectivă fără a necesita rescrierea programului. In plus dacă se lucrează cu şiruri de caractere, operațiile se repetă pentru fiecare element al şirului, iar pentru adresare reîncărcarea registrului cu deplasamentul elementului curent se poate face utilizând un mod de adresare care facilitează această operație.

Modurile complexe de adresare necesită cicluri suplimentare de memorie astfel:
- adresarea indirectă necesită un ciclu maşină pentru aflarea operandului a cărui adresă se cunoaşte
- existența deplasamentului în codul instrucțiunii necesită un ciclu suplimentar de adunare.

4.2. Moduri de adresare

Sunt prezentate in continuare cele mai utilizate moduri de adresare si exemple de folosire a lor.
Se vor folosi următoarele convenții de notare:
r = registru
d = deplasament
R(r) = conținutul registrului r
M(x) = funcție de adresare indirectă a valorii x
x sh y = delasare la stânga a lui x cu y poziții binare.
o = operand

4.2.1. Adresare imediată

În instructiune apare valoarea operandului
o = d (valoarea operandului se află în câmpul deplasament al instrucțiunii)
Exemple:

```
.data
var1    db      2
```

```
var2    dw      5 dup('ab')
.code

mov al,2                        ; valoarea operandului este în instructiune
mov ax,offset var2   ;valoarea operandului este adresa relativă in segmentul de
date
mov ax,@data                    ;valoarea operandului este adresa de segment pentru
data
```

4.2.2. Adresare directă prin registru

În instrucțiune apare adresa operandului care se încarcă în/dintr-un registru
o = R(r) (valoarea operandului se află în registru)

Exemple:

```
mov ax,var2  ; var2 reprezinta adresa de memorie de la care se incarca
operandul
mov ax, word ptr var2+1     ; word ptr specifica dimensiunea word (2 octeti)
                            pentru operand, deci se ;incarca in ax valorile de la adresa
                            var2+1
mov var2+2,ax
mov var2(2),ax
add cx,[100]
```

4.2.3. Adresare indirecta prin registru

Operandul se gaseste in registru (bx, si sau di)
Registrul conține adresa operandului.
o = M(R(r)) (adresa operandului se află în registru)

Exemple:

```
mov ax,[bx]   ;operandul se afla la adresa indicata de registrul bx
mov [di],cx
add byte ptr[si],2 ; instructiune echivalenta cu
        add[si],2       ;(cu obs. ca nu se cunoaste dimensiunea operandului)
```

4.2.4. Adresare cu autoincrementare/autodecrementare

Adresare utilizată cu registrul contor de instrucțiuni (IP) şi registrul indicator de stivă (SP).
Registrul conține adresa operandului şi valoarea acestuia se postincrementează sau predecrementează cu *n*, unde n reprezintă un număr de octeți corespunzător operației.
o = M(R(r)), R(r) = R(r)+n
R(r) = R(r)-n, o = M(R(r))

4.2.5. Adresare indirectă cu autoincrementare/autodecrementare

Un registru păstrează adresa adresei operandului, iar conținutul registrului se autoincrementează/autodecrementează:

o =M(M(R(r))), R(r) = R(r)+n
R(r) = R(r)-n, M(M(R(r)))
Adresa se află în câmpul instrucțiunii şi nu se modifică în timplul execuției. Se pot astfel referi sirurile stocate în memorie. Modul de adresare se utilizează în instrucțiunile cu şiruri.

4.2.6. Adresare indirectă bazată cu deplasament

Adresa operandului se calculează prin adunarea conținutului unui registru de bază specificat cu valoarea unui deplasament.
o =M(R(r)+d)
Acest mod de adresare este potrivit pentru adresarea elementelor unei structuri de date.
Registrul conține adresa de bază de început a structurii de date, iar deplasamentul numărul de octeți până la elementul respectiv.

Exemple:
Pentru obtinerea adresei operandului se foloseste un registru de baza (bx sau bp), iar adresa stocata in acesta se aduna cu un deplasament. Registrul segment pentru bx este ds, iar pentru bp este ss.

```
.data
var     dw      10 dup(10)
.code
mov bx,5       ; s-a pregatit registrul de baza

mov ax,var[bx]
mov ax,bx[var]
mov ax,[bx+var]
mov ax,[bx].var
;instructiunile sunt echivalente si transfera in ax al 5-lea element de doi octeti de la adresa var

sau
mov bx,offset var     ; s-a pregatit registrul de baza
mov ax,[bx]
```

4.2.7. Adresare dublu indirectă bazată cu deplasament

Adresa operandului se calculează prin adresarea locatiei indicate prin adunarea conținutului unui registru de bază specificat cu valoarea unui deplasament.
o =M(M(R(r)+d))
Acest mod de adresare este potrivit pentru adresarea variabilelor la care referirea se face printr-un pointer. Structura de date este stocată la o adresă relativă de memorie. Registrul conține adresa de bază de început a structurii de date, iar deplasamentul numărul de octeți până la elementul respectiv.

4.2.8. Adresare indirectă bazată indexată

Adresa operandului se calculează prin adunarea adresei de bază dintr-un registru cu indexul elementului ce reprezintă operandul. Indexul se obține prin shiftare la stânga cu numărul ce reprezintă puterea lui 2 necesară pentru a specifica dimensiunea unui element.

$o = M(R(r_1)+R(r_2).sh.n)$

În instructiune apar registri de baza si index utilizati pentru obtinerea adresei operandului. Adresa efectiva se obtine adunind valoarea din registrul de baza cu cea din registrul index si adresa relativa continuta in instructiune.

Exemple:

```
mov ax,var[bx][si]
mov ax,var[bx+si]
```

4.2.9. Adresare indirectă bazată indexată cu autoincrementare/decrementare

Adresa operandului se calculează prin adunarea adresei de bază dintr-un registru cu indexul elementului ce reprezintă operandul. Indexul se obține prin shiftare la stânga cu numărul ce reprezintă puterea lui 2 necesară pentru a specifica dimensiunea unui element. Conținutul registrului de bază se autoincrementează/autodecrementează.

$o = M(R(r_1)+R(r_2).sh.n),\ R(r_1) = R(r_1)+N$

4.2.10. Adresare indirectă bazată indexată cu deplasament

Adresa operandului se calculează prin adunarea adresei de bază dintr-un registru cu un deplasament şi cu indexul elementului ce reprezintă operandul. Indexul se obține prin shiftare la stânga cu numărul ce reprezintă puterea lui 2 necesară pentru a specifica dimensiunea unui element.

$o = M(R(r_1)+d+R(r_2).sh.n)$

Ne putem imagina o adresă de structură de date stocată la o adresă relativă şi care conține un şir de elemente la un deplasament *d* de începutul structurii.

Exemplu:

```
mov ax,[bx+di+offset var]
```

4.2.11. Adresare indirectă indirect bazată indexată cu deplasament

Adresa operandului se calculează indirectarea adresei obținute prin adunarea adresei de bază dintr-un registru cu un deplasament şi adunate cu indexul elementului ce reprezintă operandul. Indexul se obține prin shiftare la stânga cu numărul ce reprezintă puterea lui 2 necesară pentru a specifica dimensiunea unui element.

$o = M(M(R(r_1)+d)+R(r_2).sh.n)$

4.2.12. Adresare indirectă cu deplasament indirect bazată cu deplasament

Adresa operandului se calculează indirectarea adresei obținute prin adunarea adresei de bază dintr-un registru cu un deplasament şi adunate cu deplasamentul elementului ce reprezintă operandul.

$o = M(M(R(r_1)+d_1)+d_2)$

4.2.13. Adresare indirectă cu deplasament indirect bazată indexată cu deplasament

Adresa operandului se calculează indirectarea adresei obținute prin adunarea adresei de bază dintr-un registru cu un deplasament adunată cu deplasamentul structurii în care se află elementul ce reprezintă operandul.

$o = M(M(R(r_1)+d_1)+d_{2+} R(r_2).sh.n)$

4.3. Concluzii referitoare la modurile de adresare

Modurile de adresare pot fi împărțite în următoarele categorii:

- moduri simple
- moduri complexe (cu deplasament, registru de bază, registru index)
- moduri complexe cu autoincrementare/autodecrementare.

Cum adresarea memoriei nu se realizează static, ci dinamic, modurile de adresare complexe se potrivesc sistemelor moderne de operare şi tehnicilor dinamice de programare.

Autoincrementarea/autodecrementarea se utilizează pentru:

- extragerea instrucțiunilor de executat de către microprocesor
- lucrul cu stiva (organizată LIFO)
- operații cu şiruri

Modurile complexe de adresare scad viteza de execuție, deoarece necesită cicluri suplimentare de memorie, dar oferă flexibilitate sistemului şi programatorului.

5. Arhitecturi RISC.

5.1. Caracteristici RISC

Primele concepte au fost enuntate de Seymour Cray, cu realizarea calculatorului CDC6600:

- set simplu de instructiuni
- executie in conducta a instructiunilor
- numar mare de registri.

Termenul RISC – introdus in 1980 de Patterson&Dietzel cu realizarea unui cip microprocesor fara interpretor (unitate de comanda cablata, nu microprogramata), numit RISCI, apoi RISCII.

Caracteristici ale arhitecturilor RISC:

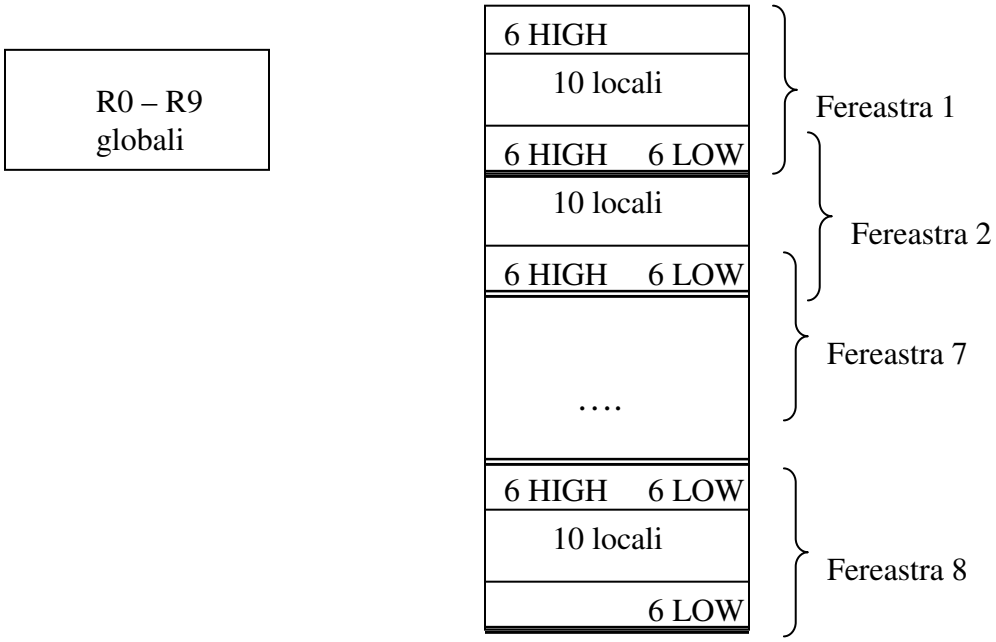
- set simplu de instructiuni, ce implementeaza numai operatiile frecvente si simple si ofera suport HLL prin alegerea judicioasa a instructiunilor.
- structura in conducta (pipeline) pentru executia instructiunilor in paralel
- numar mare de registri – operatii frecvente reg-reg
    - organizare speciala a acestora intr-un fisier de registri numit *register window*.
- numar mic de moduri de adresare
- format unic al instructiunilor (codificare)
- executia instructiunilor intr-un ciclu masina
- unitate de comanda cablata
- suport pentru intreruperi
- operatii cu memoria de tip *load* si *store*

Criteriile care stau la baza setului de instructiuni (propuse de Wulf):

- regularitatea – orice instructiune foloseste o resursa in acelasi fel
- ortogonalitatea – orice instructiune poate folosi orice mod de adresare si orice tip de data
- existenta instructiunilor primitive

Organizarea register window pentru sistemul Berkeley cu arhitectura RISC

- 138 registri pe 32 biti (R0 – R137) din care:
    - R0-R9 – registri globali
    - 32 registri pentru fiecare procedura ce formraza o *fereastra*, din care:
        - 10 registri globali
        - 6 registri *high*
        - 10 registri locali
        - 6 registri *low*
    - 8 ferestre corespunzatoare a 8 proceduri. Organizarea este circulara.



Avantaje RISC:

- set mic de instructiuni => **unitate de comanda mica** =>
  - viteza mare de executie (trasee mici)
  - decodificare rapida (putine instructiuni si unitate mica)
  - spatiu pentru numar mare de registri
  - costuri de proiectare mici date de:
    - timpul scurt necesar proiectarii
    - numar mic de erori => detectie simpla a acestora
  - scade riscul perimarii
- numar mare registri =>
  - viteza mare de calcul (operatii reg-reg rapide)
  - aranjament register window care faciliteaza transferul rapid de parametri
  - ofera suport compilatorului de optimizare a codului
- format unic de instructiuni- faciliteaza manipularea uniformizata a fluxului de instructiuni in conducta de executie.

Dezavantaje RISC:

- numar mic de instructiuni => creste codul scris pentru implementarea instructiunilor complexe => programe lungi => sarcini suplimantare pentru programator si masina.

5.2. UltraSPARC II

Microarhitectura

- Masina pe 64 biti (magistrala interna si de date)
- 1285 adrese

- arhitectura RISC
- instructiunile – orientate pentru executie in banda de asamblare intr-un ciclu (au 2 registre sursa si unul destinatie)
- arhitectura superscalara – 4 instrunctiuni / ciclu de ceas
- exista suport hardware pentru incarcari speculative sub forma de instructiuni PREFETCH care nu genereaza eroare (pentru a avea datele necesare la executie)

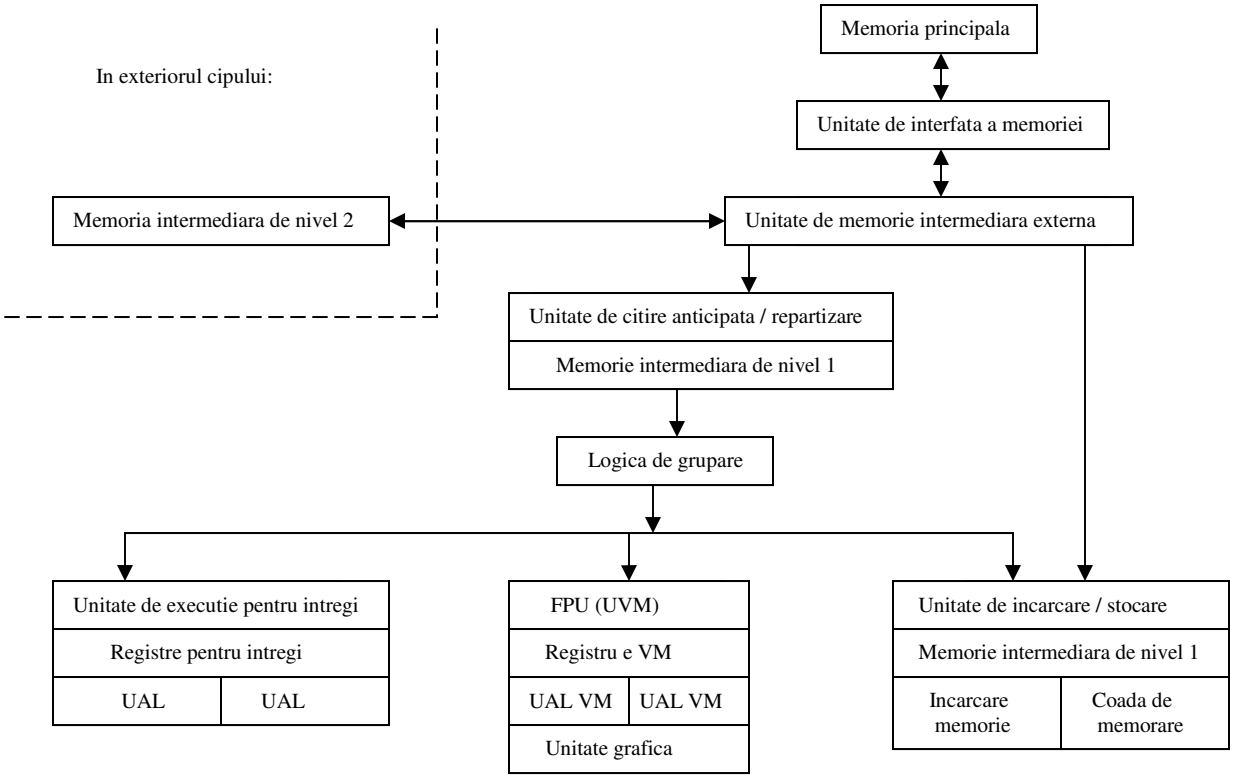


Fig. Microarhitectura UltraSPARC II

Unitatea de memorie intermediara externa gestioneaza rotarile in memoria intermediară de nivel 1, cautand linia dorita in memoria intermediara externa.

Unitatea de citire anticipata / repartizare = Unitate de Citire / Decodificare cu avantajul ca instructiunile nu se divizeaza in microinstructiuni (sunt deja microoperatii). Cuprinde un predictor de instructiuni.

Logica de grupare – selecteaza 4 instructiuni din coada pentru lansare in executie care trebuie sa poata fi executate simultan (2 UAL – intregi , 2 UAL – UM)

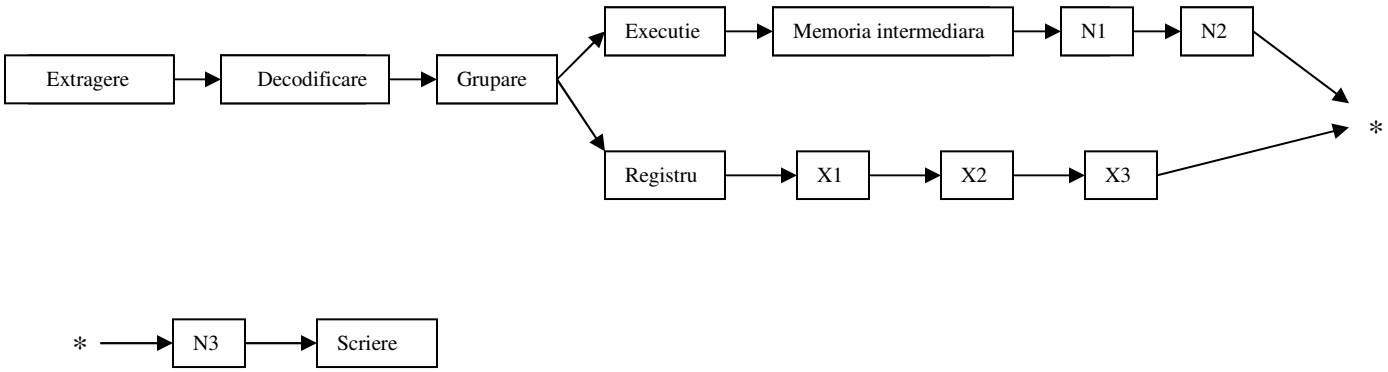
Unitatile aritmetice – (UAL) intregi si in UM contin propriile registre.

Unitatea grafica – poate executa instructiuni speciale MMX.

Unitatea de Incarcare / Memorare – gestioneaza instructiiunile LOAD / STORE pentru care exista cozi separate.

Banda de asamblare

- are 9 segmente – diferite pentru intructiuni: intregi, VM (virgulă mobilă)



Extragere – citeste instructiunea din memoria interna (fara ratari in memoria intermediara, fara predictii gresite, fara intructiuni dificile, combinarea instructiunilor corecte, etc) poate furniza4 instructiuni / ciclu la infinit.

Segmentul de Decodificare – adauga biti suplimentari fiecarei instructiuni inainte de a o pune in coada de instructiuni, pentru a accelera prelucrarea ulterioara (de exemplu – directionarea imediata spre unitatea de executie adecvata).

Segmentul de grupare – corespunzator unitatii logice de grupare – examinand instructiunile decodificate pe grupuri de 4 segmente:

- N1, N2 – nu executa nimic pentru instructiunile cu registre → sunt incluse pentru sincronizarea benzilor de asamblare.

Pentru intregi executia se face intr-un ciclu in segmente de executie.
Pentru VM executia se face in cele 4 segmente ale benzii de asamblare:

- segmentul 1 (registru) e folosit pentru accesul la registrele VM;
- urmatoarele 3 segmente pentru executie.

Segmentul N3 – comun ambelor unitati – folosit pentru a rezolva conditiile de exceptie.
Segmentul Scriere – pentru scrierea in registre.

ISA – la SPARC II

- memoria adresabila – 64 biti – adrese
- ordine octeti – big endian (c.m.s. intai) dar se poate schimba – prin setare unui bit in PSW.
- are 2 grupuri de registre: - 32 registre de uz general pe 64 biti (RØ÷R31);
  - 32 registre pentru VM.

Registre	**Alt nume**	**Functii**
RØ	GØ	- fixat hardware la Ø
R1 – R7	G1 – G7	- folosite pentru variabile globale
R8 – R13	08 – 013	- folosite pentru parametrii procedurilor apelate

R14	SP	- indicator stiva
R15	07	- registru de lucru
R16 – R23	L0 – L7	- pastreaza variabilele locale ale procedurii curente
R24 – R29	IØ – I5	- pastreaza parametrii de intrare
R30	FP	- indicatorul bazei cadrului de stiva curent
R31	I7	- pastreaza adresa de intoarcere a procedurii curente

Organizare registre: Register Window → organizare ce permite lucrul cu 32 registre la un moment dat din mai multe seturi de registre.

Exista registrul CWP (Current Window Pointer) – indicator al ferestrei curente.

6. SUBSISTEMUL DE INTRARE/IEŞIRE

1. Magistrale – conecteaza unitatile functionale interne ale sistemului de calcul
2. Sistemul de I/E – conecteaza dispozitivele de I/E

6.1. Magistrale

Magistrala este o cale electrica intre mai multe unitati functionale.
Se clasifica in:

- magistrale interne – in interiorul microprocesorului
- magistrale externe – in exteriorul microprocesorului

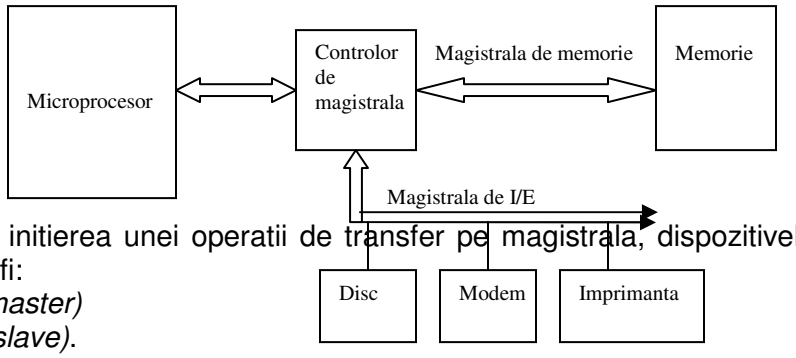
Pentru o magistrala se defineste un *protocol de magistrala (bus protocol)* care precizeaza regulile de functionare pe magistrala, anume:

- semnificatia semnalelor (date, adrese, control si stare)
- nivelele electrice ale semnalelor
- specificatii mecanice si electrice

Primele calculatoare aveau o singura magistrala care trecea pe la toate componentele sistemului de calcul numita *magistrala sistem* (system bus).
Calculatoarele noi au:

- o magistrala specializata intre microprocesor si memorie numita *magistrala de memorie*
- una sau mai multe magistrale ce leaga microprocesorul cu dispozitivele de I/E, numite *magistrale de I/E*.

Functionarea magistralelor
Gestiunea magistralei este realizata de un *controlor de magistrala* (bus controller)



In raport cu initierea unei operatii de transfer pe magistrala, dispozitivele conectate la aceasta pot fi:

- *active (master)*
- *pasive (slave)*.

Relatia dintre ele se numeste master-slave. Masterul da comenzi (activ), iar slave-ul le executa (pasiv).
Memoria are rol de slave. Celelalte dispozitive pot fi master sau slave.

Exemple de relatii master-slave:

Master	Slave	Operatie
UCP	Memoria	Prelucrarea instructiunilor si datelor
UCP	Dispozitiv de I/E	Initializarea transferurilor de I/E
UCP	Coprocesor	UCP trimite instructiuni coprocesorului
Coprocesor	UCP	Coprocesorul prelucreaza operanzii de la UCP

I/E Memorie Acces direct la memorie (DMA)

Legatura la magistrala se face prin circuite specializate de amplificare a semnalelor: *bus driver* (pentru dispozitivele master), *bus receiver* (pentru dispozitivele slave), *bus transceiver* (pentru dispozitivele cu dublu rol).

Performantele unei magistrale sunt legate de *viteza de transfer* si de *largimea de banda*. Acestea sunt determinate de principalii parametri de proiectare, anume:

- numarul de linii de adresa, de date si de control
- frecventa ceasului
- mecanismul de arbitraj
- gestionarea intreruperilor
- gestionarea erorilor

} Tipul magistralei

L*argimea de banda* este proportionala cu cantitatea de informatie transferata pe magistrala in unitatea de timp.

In functie de sincronizarea operatiilor de transfer magistralele se impart in:

- magistrale sincrone
- magistrale asincrone

Magistrale sincrone

Transferul datelor se face sincronizat dupa un semnal de deas magistrala. Frecventa lui determina lungimea unui *ciclu magistrala*. Fiecare activitate se executa intr-un numar intreg de cicluri magistrala. De exemplu o operatie de R/W se executa in cel putin trei cicli magistrala (timpi pentru stabilire adrese, semnale de comanda si date propriu zise).

O varianta de accelerare utilizeaza transferurile sincrone in blocuri de date (burst), indicandu-se adresa initiala si numarul de octeti de transferat.

Magistrale asincrone

Nu exista un ceas master, iar ciclurile de magistrala pot avea orice lungime. Protocolul este format dintr-un set de semnale care se interblocheaza (handshaking). Se utilizeaza semnale suplimentare pentru realizarea unui dialog cu interblocare intre cele 2 dispozitive, master si slave, intre care se face transferul. Dialogul este independent de timp (semnalele se comanda reciproc), viteza de transfer adaptandu-se astfel automat la performantele dispozitivelor aflate in dialog.

Arbitrajul magistralei

Deoarece atat procesorul cat si adaptoarele dispozitivelor de I/E pot dori simultan sa devina master pe magistrala, este necesara arbitrarea cererilor acestor dispozitive active.

Arbitrarea magistralei se poate realiza:

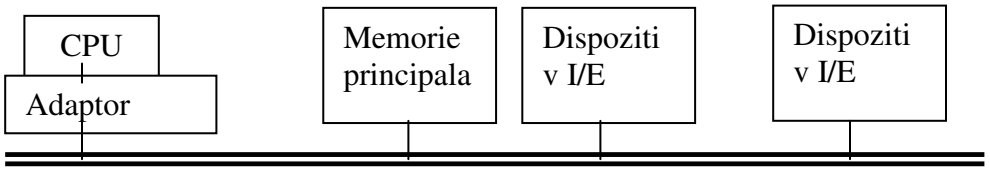
- centralizat (cu dispozitiv specializat ca arbitru de magistrala - unele microprocesoare pot avea incorporat un astfel de dispozitiv)
- descentralizat, cu linii suplimentare pe magistrala

Ambele variante asigura accesul la magistrala functie de *prioritatile* alocate initial dispozitivelor aflate in conflict.

Standarde de magistrala

ISA (Industry Standard Architecture)

Setul de chip-uri ce implementeaza *controller-ul de magistrala* este conectat direct la magistrala interna a microprocesorului



Suporta pana la 32 biti de date.
Obs. Largimea de banda reala este 1 / 2 din cea teoretica, datorita protocoalelor de comunicatie conform carora se face transferul pe magistrala de I/E.
Standardul ISA ofera suport DMA.
Adaptorul DMA este un dispozitiv programabil ce poate efectua transferuri intre MI si dispozitivele periferice, in paralel cu functionarea CPU. In vederea initierii unui transfer direct, procesorul va incarca in registrele adaptorului DMA numarul dispozitivului I/E, adresa de memorie, numarul de octeti de transferat si sensul transferului. Pentru a indica incheierea operatiilor comandate, adaptorul DMA va lansa o intrerupere catre procesor.

MCA (MicroChannel Architecture) este un standard pe 32 biti.Arhitectura este complet diferita si net superioara celei anterioare. A fost lansat de IBM in cadrul politicii de promovare pe piata a seriei de calculatoare PS/2.

EISA (Extended ISA)
A fost creat ca reactie a restului industriei de calculatoare personale la MCA.
Este o arhitectura pe 32 biti si care inglobeaza facilitatile necesare conectarii de procesoare multiple. Asigura, de asemenea o legatura mai rapida cu hard-discul (care poate avea fie adaptor EISA, fie poate fi conectat prin adaptor SCSI).

VLB (VESA- Video Electronics Standard Association- Local Bus)
In acest caz, o parte din adaptoarele de I/E au acces la vitezele sporite ale magistralei procesorului.
Este organizata pe 32 biti si ofera o rata maxima de transfer de 128-132 Moct/s.

PCI (Peripherial Component Interconnect Bus)
Este o alternativa la VLB, dar presupune intercalarea unor punti (bridge) intre magistrala procesorului si magistrala de I/E. Datorita arhitecturii sale se mai numeste si *magistrala mezanin*. Transferul se realizeaza mai rapid, deoarece magistrala PCI lucreaza in paralel cu magistrala procesorului, fara sa o inlocuiasca (ex. In timp ce se executa un transfer CPU-Cache, pe PCI se pot executa alte transferuri).

PCMCIA (Personal Computer Memory Card International Association)
A fost creata pentru calculatoarele portabile, ce contin placi de dimensiuni reduse.

AGP (Accelerated Graphics Port) este o magistrala care faciliteaza capabilitati grafice de performanta, in special 3D. Aplicatiile 3D cer pe langa spatii mari de stocare a datelor necesare operatiei de refresh a monitorului si spatii mari de stocare pentru formatele speciale de date 3D (z-buffering, alpha blending, texture mapping). Standardul permite accelerarea aplicatiilor 3D.
AGP adauga caracteristici noi acceleratoarelor grafice ca acces in conducta dedicat (dedicated pipelined access) la memoria primara si viteze mari de transfer, furnizand astfel largime mare de banda.
Magistrala AGP nu inlocuieste magistrala PCI, ea este proiectata special pentru componente grafice punct la punct. Ea este separata de magistrala PCI si utilizeaza un conector separat.

USB (Universal Serial Bus) –este un standard de magistrala modern (1990) introdus pentru conectarea externa a dispozitivelor de viteza redusa (tastatura, mouse, scanner) la magistrala microprocesorului printr-un hub central (root hub). Hubul are socluri pentru dispozitivele de I/E sau la huburi de extindere. Astfel ca topologia unui sistem USB este un arbore cu radacina la hubul central.

Standardele de magistrala prezentate sunt utilizate la calculatoarele personale (PC-uri). Pentru calculatoarele de tip **mainframe**, utilizate la sistemele mari, industriale, se utilizeaza standardul VME.

Magistrala VME (Versa Module Eurocard)
Este derivata din magistrala de tip **backplane** utilizata la sistemele bazate pe microprocesoare Motorola.
Obs. La calculatoarele compatibile IBM PC adaptorul de magistrala se afla pe placa de baza (motherboard). La mainframe-uri se utilizeaza **backplane-ul** care nu contine circuite active, iar adaptorul de magistrala se afla pe o placa speciala, conectata la backplane, ca toate celelalte placi din sistem.
Ulterior a fost adaptata la formatul de placi Eurocard.
Standardul este foarte bine definit si documentat, astfel incat orice numar de componete proiectate conform acestuia pot lucra impreuna.
Obiectivele urmarite la proiectarea acestui standard au fost:
- interoperabilitate – asigurata prin modul de definire a standardului
- performanta ridicata - deoarece este **asincrona** si se adapteaza vitezelor dispozitivelor implicate in transfer
- siguranta ridicata - oferita prin proiectare mecanica dar si prin protocolul de transfer.

Arhitectura magistralei VME ne prezinta o familie de 3 magistrale:
- VME – magistrala ce leaga procesorul, memoria si sistemul de I/E
- VSB – leaga procesorul cu memoria locala
- VMS – magistrala seriala pentru comunicatiile si semnalele de sincronizare intre mai multe procesoare.

Pentru fiecare transfer se specifica adresa si dimensiunea datelor astfel incat diferite perechi master/slave pot comunica pe acceasi magistrala in grupuri de biti (8, 16, 32) de dimensiuni optime.

Exista suport pentru sistemele cu procesoare multiple. (ciclu read-modify-write indivizibil, posibilitatea de a instala mai multe controller-e pentru gestionarea intreruperilor).
Exista un dispozitiv *bus timer* absolut necesar la magistralele asincrone, care are rolul de a monitoriza magistrala in vederea depistarii si solutionarii blocajelor aparute in cursul unor dialoguri master/slave.

6.2. Sistemul de intrare/ieşire

Sistemul de I/E este componenta sistemului de calcul ce realizeaza comunicarea acestuia cu mediul extern.

Fiecare echipament de I/E e insotit atat de:
1. *interfata hardware (**controller sau adaptor)*** formata dintr-o serie de componente electronice ce asigura compatibilizarea nivelelor de semnal cu standardele de magistrala utilizate, precum si o prima prelucrare logica a acestora.

2. *interfata software (**driver)*** ce asigura o prelucrare logica si compatibilizarea semnificatiei semnalelor de la iesirea interfetei hard cu standardele soft ale sistemului de operare.

Fiecare adaptor al unui dispozitiv de I/E contine (cel putin) 2 registrii. Unul va fi utilizat pentru transferul datelor, iar celalalt va contine informatii de comanda si stare. O informatie de stare esentiala este conditia *ready*, care arata daca adaptorul a incheiat operatia comandata si este pregatit sa preia o noua comanda. O operatie de I/E se executa prin transferul informatiilor de control si stare precum si a datelor propriu-zise intre procesor si registrele adaptorului. Transferul datelor intre procesor si adaptor are loc daca adaptorul este *ready*, adica a incheiat o operatie de intrare si are date pregatite pentru transfer in registrul de date, sau a incheiat o operatie de iesire si poate accepta alte date in registrul de date.
Exista doua categorii de arhitecturi ale sistemelor de I/E utilizate la sistemele de calcul moderne:
- canale de date
- DMA

Canale de date

Canalele de date se utilizeaza in special la calculatoarele de tip *mainframe*, cu arhitecturi complexe, conectate in retele cu utilizatori multiplii, frecventa operatiilor de I/E este foarte mare, iar magistrala foarte solicitata. La aceste sisteme se utilizeaza calculatoare specializate pe realizarea transferurilor de I/E, numite *canale de I/E*. Arhitectura unui sistem cu canale de I/E presupune existenta unor magistrale specializate, independente:
- o magistrala pentru accesul CPU la memorie,
- magistrala de I/E prin care CPU comunica cu canalele,
- magistrala memoriei prin care canalele comunica cu memoria.

Un canal de date este de fapt un *procesor de I/E* la care sunt atasate toate dispozitivele de I/E. Acesta va executa un *program* special, comunicat de CPU si destinat executarii unui transfer complex. Se va executa urmatoarea succesiune de operatii:
- CPU transmite canalului programul de I/E .
- Canalul executa operatiile, realizand transferuri directe intre memoria principala si sistemul de I/E.
- Canalul semnalizeaza, printr-o intrerupere la procesor, incheierea transferului.

Functie de viteza de transfer a dispozitivelor atasate, exista doua tipuri de canale de I/E:
- canal *multiplexor,* la care se atasaza dispozitive lente (imprimante, terminale), canal care realizeaza, in paralel, operatii din programe de transfer pentru mai multe din dispozitivele conectate la el.
- canal *selector*, la care se atasaza dispozitivele rapide (discuri) si care realizeaza un singur transfer la un moment dat.

Avantajul acestei organizari consta in transferarea de la CPU catre procesorul de I/E a responsabilitatii majoritatii operatiilor legate de transferul datelor intre periferice si memoria principala.

DMA (acces direct la memorie)

Calculatoarele personale (PC-uri) au o structura mai simpla a sistemului de I/E. Acestea sunt organizate in jurul unor magistrale la care se conecteaza principalele module ale sistemului: placa de baza (procesor, memorie, controller-e pentru dispozitivele standard mai putin placa video) si controller-ele dispozitivelor de I/E.

Adaptoarele dispozitivelor ce transfera blocuri de date (discuri) realizeaza acces direct la memorie (*DMA)*, adica transfera blocuri de caractere direct intre disc si memoria principala, fara a ocupa timpul procesorului central.
Aceasta schema este utilizata pentru perifericele rapide si presupune eliberarea CPU de detaliile de realizare a transferului propriu-zis a unui grup de informatii intre adaptorul dispozitivului de I/E si memoria principala. Aceasta functie a CPU este preluata de DMA, dispozitiv conectat la magistrala sistemului.
Chip-ul DMA va contine cel putin 4 registrii care vor fi incarcati, de catre programul ce se executa pe procesor, cu urmatoarele informatii ce definesc transferul:

- adresa de memorie la care se face transferul,
- numarul de octeti sau cuvinte ce trebuie transferat (contor),
- identificarea, in spatiul adreselor de I/E, a dispozitivului de I/E cu care se face transferul,
- sensul transferului (citire = intrare, scriere = iesire).

Utilizand aceste informatii, dispozitivul DMA va realiza transferul comandat, prin magistrala sistemului. DMA devine astfel dispozitiv activ pe magistrala, intrand in competitie cu procesorul. Pentru transferul fiecarui cuvant va emite semnalul *bus request* catre dispozitivul de arbitraj a accesului pe magistrala si doar la incheierea transferului comandat va lansa o intrerupere catre procesor.
Referitor la protocolul de acces la magistrala, DMA este prioritar fata de CPU, deoarece transferurile cu dispozitivele de I/E sunt, in general, conditionate de timp.

Problema ce apare in acest caz este legata de faptul ca aceste transferuri directe se fac pe acceasi magistrala cu restul transferurilor din sistem. Solutia problemei consta in utilizarea unui mecanism pentru *arbitrarea accesului la magistrala.*

6.3. Mecanisme de comunicație

Comunicația intre doua echipamente se realizeaza prin transmiterea semnalelor de la un echipament la celalalt. Cel care transmite se numeste *transmitator*, iar celalalt *receptor*.
Pentru transmisia la distanta se utilizeaza modem-uri.
Cele doua parti trebuie sa respecte regulile impuse de protocolul de comunicare implementat.
Exista doua tipuri de transmisie:

- sincrona
- asincrona

Este necesar un mecanism pentru realizarea sincronizarii operatiilor de transfer intre 2 calculatoare, astfel incat informatiile receptionate sa fie identice cu cele emise pe linia seriala.
Sincronizarea se poate face fie prin sincronizarea semnalelor de tact ale celor doua parti (suport hard), fie prin continutul mesajului transmis (suport soft).
Transmisiile *sincrone* necesita o operatie de sincronizare initiala si promisiunea de stabilitate a ceasurilor la cele 2 parti. Datele sunt transmise continuu, iar pentru perioadele de pauza se transmite codul unui caracter special (*idle*).
Transmisiile *asincrone* realizeaza sincronizarea la nivelul fiecarui cod (octet), atasand acestuia biti suplimentari. Intre transmisia a 2 octeti linia este pe “1” (asigura detectarea purtatoarei), primul bit va fi bitul de START (**= 0**), urmeaza un numar predefinit de biti de date si 1 sau 2 biti de STOP (**=1**) .

Moduri de transmisie

- *simplex* – exista un fir de date si transmisia datelor se face intr-un singur sens, in sens invers circuland doar semnale de confirmare (ACK)
- *semi-duplex* – un fir de date, iar transmisia se face in ambele sensuri, alternativ, conform unui *protocol*. Exista si fire separate pe care se realizeza protocolul
- *duplex* – cu fire separate de date pentru fiecare directie, iar transmisia se face in ambele sensuri simultan. Pentru protocol exista fire separate.

7. TEHNICI PENTRU CRESTEREA PERFORMATEI SISTEMULUI DE CALCUL

7.1. Performanța sistemului de calcul

Factorii care afecteaza performanta sistemului de calcul sunt:
- arhitectura CPU
- dimensiunea si metodele de implementare a setului de instructiuni
- abilitatea compilatorului de optimizare a codului (pentru creşterea performanței sistemului)
- tehnologia de realizare a cipurilor
- sistemul de operare

Performanța calculatorului poate fi descrisă cu următoarea formulă
Performanța = 1/T*p*c
unde :
T = perioada ciclului instrucțiune. Ea este invers proporțională cu frecvența ceasului sistemului de calcul. Aceasta este limitată de tehnologie. Creşterea frecvenței duce la creşterea performanței sistemului.
p = lungimea căii (path lenght) este numărul de instrucțiuni maşină necesare pentru execuția unei comenzi. Este influențată de arhitectura CPU, de setul de instrucțiuni şi de compilatorul cu optimizare. Performanța creşte cu scăderea lui p.
c = ciclii/instrucțiune reprezintă numărul de ciclii necesari pentru execuția unei instrucțiuni.
Este foarte important ca software-ul să exploateze eficient arhitecturile software. Cum acest lucru este realizat de sistemul de operare si de compilatoare care folosesc setul de instrucțiuni, pentru a obține o performață maximă acesta (setul de instrucțiuni) devine un factor critic de performanță.
Alți factori care influențează performanța sistemului sunt:
- timpii de acces la memorie
- caracteristicile memoriei externe
- rata de transfer a datelor de I/E
- comunicațiile.

Principalul descriptor al performanței = abilitatea de a executa programe scrise în limbaje de nivel înalt (HLL). Eficacitatea execuției acestor programe depinde de
- arhitectura sistemului
- tehnologia de realizare a compilatorului. Optimizarea programelor tinde să devină o componentă standard a compilatorului, dar trebuie să existe suport hardware pentru acest lucru. E necesar să se simplifice setul de instrucțiuni pentru maximizarea vizibilității tuturor operațiilor.

Arhitectura sistemului influențează costul şi performanța.

Sistemul de operare cere şi el caracteristici arhitecturale ca:
- moduri de utilizare privilegiat şi utilizator
- suport pentru întreruperi hard şi soft
- suport pentru primitivele de sincronizare în configurații multiprocesor

Organizarea hardware influențează performanțele sistemului astfel:

- viteza limitată a tehnologiei cere arhitecturi speciale – implementarea calculului paralel, de exemplu
- dimensiunea finită a cipului impune restricții în manipularea datelor
- costul complexității arhitecturii – nici o facilitate nu se obține pe gratis

7.2. Tehnici de creştere a performanței

Se poate realiza creşterea performanțelor sistemelor de calcul şi prin software şi prin hardware.
- software - metodele de creştere a performanțelor cu ajutorul compilatoarelor se numesc statice, pentru că programul este analizat şi optimizat o singură dată
- hardware – metode dinamice pentru că sunt aplicate în timp ce programul se execută.

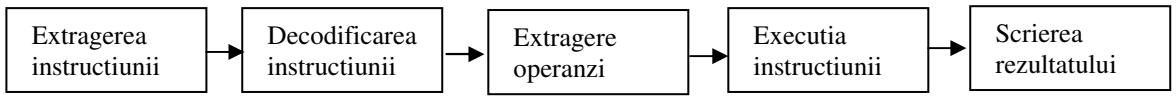
Tehnicile hardware rezultate din formula sunt::
- scurtarea căii instrucțiunii
- îmbunătățirea raportului instrucțiuni/ciclu
- creşterea frecvenței ceasului sistem.

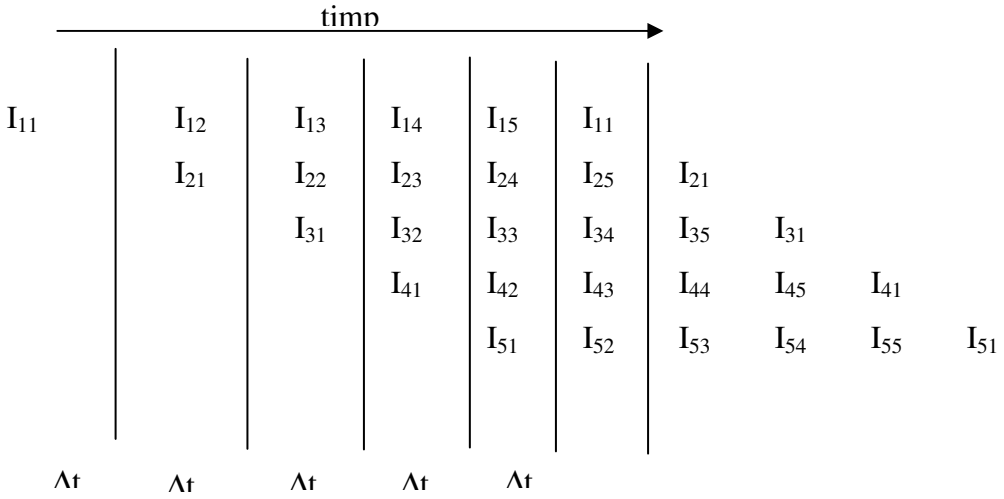
Există două tehnologii arhitecturale esențiale:
- exploatarea paralelismului la nivel instrucțiune (Instruction Level Parallelism, ILP)
- ierarhii sofisticate de memorie (cache-uri)

7.2.1. Paralelismul execuției instrucțiunilor

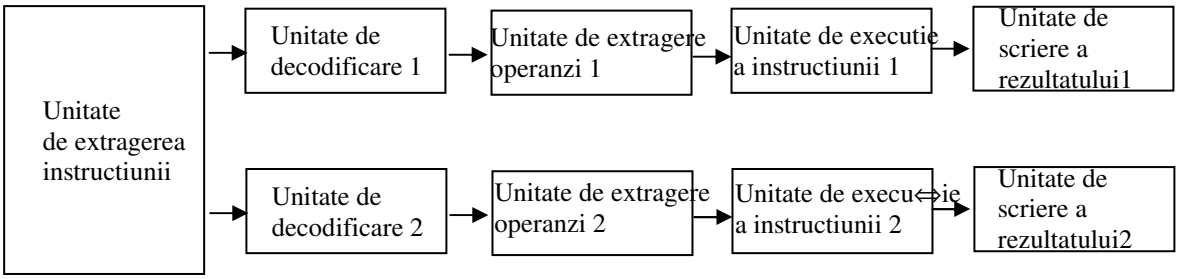
Constă în independența instrucțiunilor din programe una de cealaltă. Acest lucru permite execuția mai multor instrucțiuni simultan.
Există două forme de paralelism:
- banda de asamblare a instrucțiunilor succesive (în 1985 au apărut primele microprocesoare cu pipeline);
- execuția în paralel a instrucțiunilor independente (în 1990 au apărut primele procesoare VLIW, iar în 1995 procesoare superscalare pot executa instrucțiuni în ordini foarte diferite- out of order execution): procesoare de tip VLIW (very long instruction word) cu compilatoare care aleg instrucțiunile care se pot executa în paralel şi procesoarele superscalare care fac alegerea în timpul execuției.

Banda de asmblare (pipeline) se referă la extragerea instrucțiunilor de executat într-o zonă tampon pentru extragere în avans (prefetch buffer) a.î. atunci când este necesară instrucțiunea se cieşte din tampon, nu din memorie.
Execuția instrucțiunilor este împărțită în mai multe trepte (părți). Pentru fiecare parte existând o componentă hardware dedicată acesteia a.î. toate părțile să poată funcționa în paralel. Treptele se numesc şi segmente(stages) sau unități.

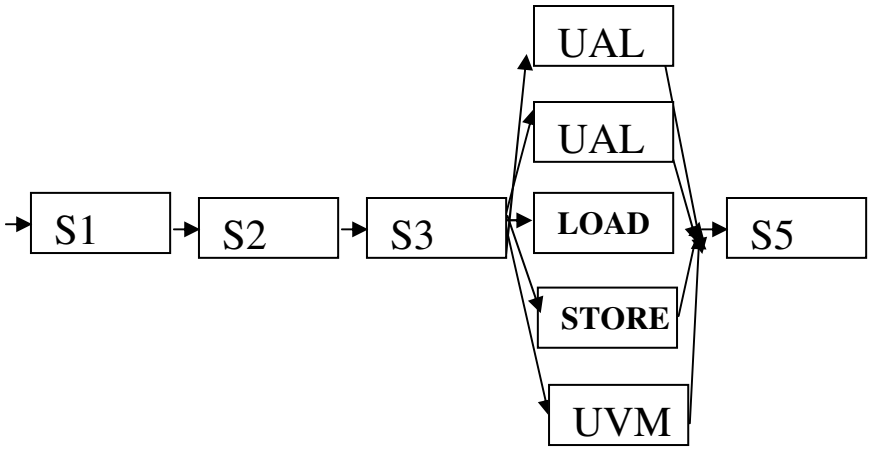




La microprocesorul Pentium II există două benzi de asamblare pentru instrucțiuni:
- una principală - pentru orice instrucțiune
- una pentru instrucțiuni simple pentru întregi.



Execuția instrucțiunilor nu realizează o creştere în viteză cu numărul de trepte deoarece multe instrucțiuni necesită rezultatul instrucțiunilor precedente pentru a putea fi executate.
Arhitecturile superscalare sunt arhitecturile cu mai multe unități de calcul puse în paralel.
Anumite segmente ale benzii de asmblare se execută mai repede decât altele. Acest lucru a dus la realizarea arhitecturii superscalare, în care mai multe unități de acelaşi fel sunt puse în paralel şi execută în paralel datele furnizate de treapta inferioara.



Generația următoare:
Perfecționarea tehnicilor de exploatare a paralelismului şi a ierarhiilor de memorii:
Trace cache – un cache pentru instrucțiuni care, în loc de a păstra instrucțiunile în ordinea adreselor lor, le menține în ordinea în care este probabil să fie executate.
Execuția speculativă şi predicția valorilor – se referă la dependențele dintre instrucțiuni , atunci când o instrucțiune are nevoie de rezultatul alteia pentru execuție. În acest caz se încearcă ghicirea valorii rezultate a instrucțiunii anterioare. Când rezultatul unei instrucțiuni soseşte se vede dacă predicția a fost corectă. Dacă da atunci instrucțiunea următoare se consideră deja executată, dacă nu se reexecută.
Predicția predicată - aceasta evită execuția instrucțiunilor de salt (care au un efect negativ asupra performanței.

8. SISTEME CU PROCESOARE MULTIPLE

8. 1. Forme de paralelism

Numim sistem multiprocesor un sistem cu mai multe unitati centrale de procesare.
Au fost introduse pentru cresterea performantelor sistemelor de calcul prin:

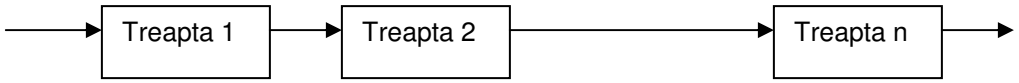
- cresterea vitezei de executie – mai multe module ale aceleiasi aplicatii se executa in paralel ca procese independente pe mai multe procesoare
- cresterea volumului lucrarilor trecute prin sistem, mai multe programe si mai multe date corespunzatoare proceselor paralele care se executa
- cresterea fiabilitatii sistemelor prin disponibilitatea mai multor procesoare in caz de defectare.

Forme de paralelism
Exista doua forme extreme de paralelism:

1. conducta (pipeline)
2. matricea paralela (parallel array)

1. Conducta (pipeline)

Conducta este alcatuita din mai multe trepte de prelucrare fiecare continand un registru de intrare si o logica combinationala de care produce iesirea.

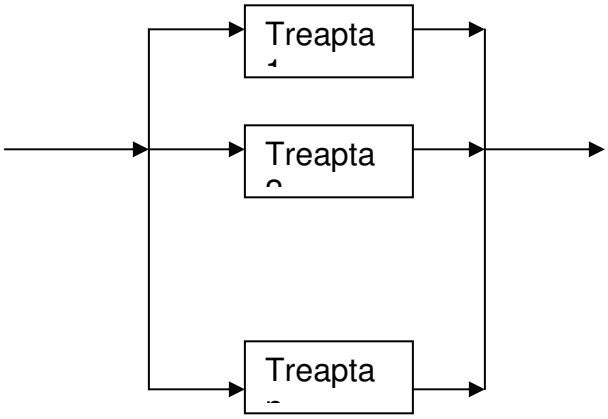


Fiecare treapta introduce o intarziere. Dupa scurgerea celei mai lungi intarzieri, un ceas determina transferul iesirilor treptelor in registrele de intrare ale treptei urmatoare.
La fiecare perioada a ceasului se produce un rezultat. O conducta cu *n* trepte produce *n* rezultate. Se poate astfel obtine un spor de performanta egal cu numarul treptelor. Performanta scade cand o treapta ramane inactiva sau livreaza prea incet rezultatul.
Tehnica este utilizata pentru :

- implementarea paralelismului instructiunilor in UCP-urile microprocesoarelor actuale
- pentru realizarea procesoarelor vectoriale in care un flux de date este prelucrat de mai multe UCP

2. Matricea paralela (parallel array)

Matricea paralela este alcatuita din mai multe unitati functionale individuale care executa sarcini complexe in unitatea de timp prelucrand fiecare alt flux de date. Daca toate unitatile sunt ocupate se obtine un spor de viteza egal cu numarul unitatilor.



8.2. Clasificarea lui Flynn

Fluxurile care interactioneaza intr-un sistem de calcul sunt:

- Fluxul de instructiuni (instruction stream) care furnizeaza comenzi logicii de prelucrare a datelor
- Fluxul de date (data stream) care furnizeaza datele necesare in prelucrare (operanzii)

Flynn a clasificat sistemele cu procesoare in patru clase in functie de fluxurile care interactioneaza in:

1. *SISD (Single Instruction Single Data)*

Sistemele din aceasta clasa au un unic flux de instructiuni si un unic flux de date.
Ele au o unitate de comanda care emite comenzile fluxului de instructiuni care prelucreaza fluxul de date.
In aceasta clasa intra sistemele monoprocesor cu arhitectura von Neumann.

2. *SIMD (Single Instruction Multiple Data)*

Sistemele au un singur flux de instructiuni care prelucreaza mai multe fluxuri diferite de date. Pe fiecare traseu de date se executa aceleasi instructiuni.
In aceasta clasa intra sistemele multiprocesor care implementeaza matricea paralela.
Sistemele SIMD sunt utilizate in aplicatii ingineresti care prelucreaza un volum mare de date (prelucrarea imaginilor de exemplu) si care utilizeaza calcule cu matrici, solutii ale sistemelor liniare, transformate Fourier.

3. *MISD (Multiple Instruction Single Data)*

In aceasta clasa intra sistemele cu multiple fluxuri de instructiuni ce prelucreaza acelasi flux de date. Sistemele implementeaza conducta ca forma de paralelism si se numesc *sisteme vectoriale*. Ele sunt utilizate in aplicatiile care realizeaza un numar mare de operatii asupra unui set mic de date.

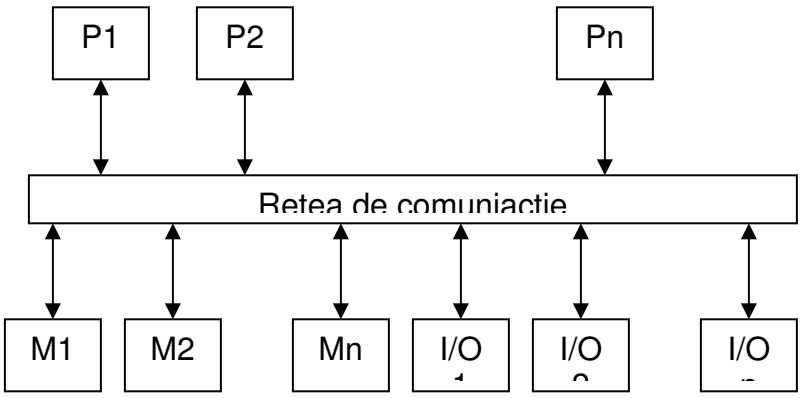
4. *MIMD (Multiple Instruction Multiple Data)*

Sistemele din acesta clasa au mai multe fluxuri de instructiunui ce prelucreaza mai multe fluxuri de date. Arhitectura MIMD contine mai multe UCP ce executa simultan diverse secvente de instructiuni pentru date diferite.
Sistemele MIMD au fost clasificate de Enslow in doua clase in functie de modul de conectare a resurselor la sistemul de calcul astfel:

a) *MIMD puternic cuplate*
b) *MIMD slab cuplate*

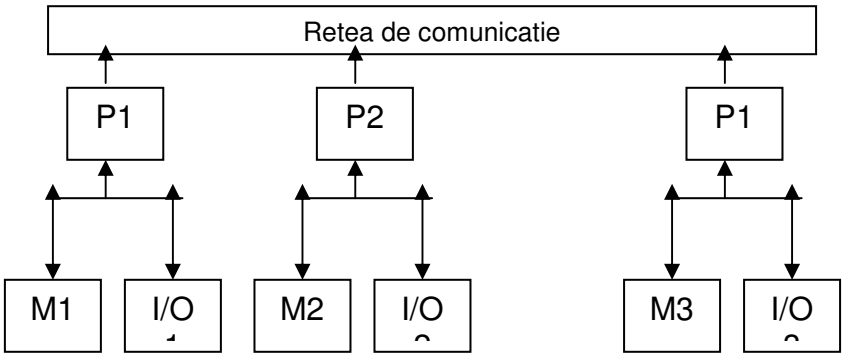
a) *MIMD puternic cuplate*

In aceste sisteme resursele sunt conectate separat la reteaua de comunicatie si sunt partajate de toate procesoarele.



Resursele comune sunt accesibile tuturor procesoarelor si de aceea apar conflicte la accesul lor si la validarea accesului la date. Sunt necesare tehnici speciale de sincronizare pe magistrala precum si de planificare a activitatii procesoarelor.

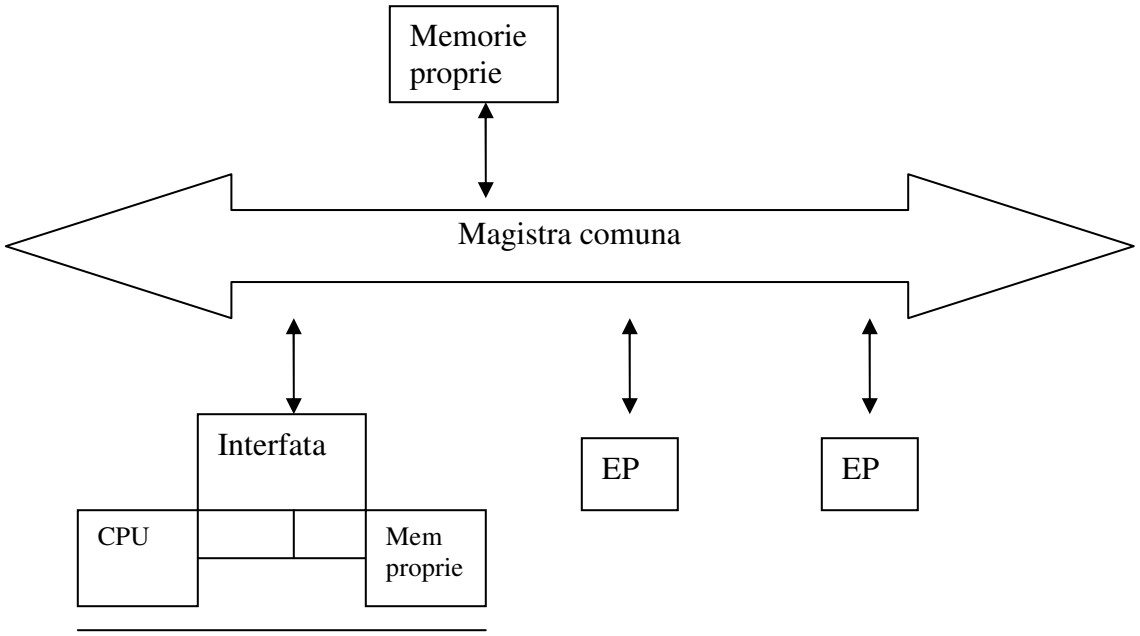
b) MIMD slab cuplate
Sunt sisteme multiprocesor cu resurse locale identificate ca *sisteme distribuite*. Ele sunt configurate ca sisteme independente de calcul unicul mod de utilizare in comun a acestora fiind transmisia. Timpul de obtinere a datelor depinde de distanta dintre procesoare, deci reteaua de comunicare joaca un rol foarte important in constructia acestora.



8.3. Organizarea memoriei

Sistemele multiprocesor se clasifica in functie de organizarea memoriei in
1. Sisteme multiprocesor cu memorie utilizata in comun
2. Sisteme multiprocesor cu memorie locala(proprie)

1. Sistemul multiprocesor cu memorie comuna foloseste o mem globala accesibila tuturor procesoarelor
   - toate procesoarele sunt active, avand acces la acelasi front de lucru
   - apar conflicte la accesul memoriei si al validarea accesului la date
   - nu sunt folosite in sistemele mari



Folosirea in comun a magistralei precum si a memoriei comune impune folosirea de tehnici speciale de sincronizare pe magistrala, precum si de planificare a activitatii procesoarelor.
Ca tehnica de sincronizare la structura multiprocesor cu magistrala comuna se aminteste tehnica TAS(Test And Set).
Similara celei folosite in sistemul monoprocesor, dar care foloseste linii speciale de stare a magistralei(ocupata sau nu) precum si cicluri speciale citire-modificare-inscriere, diferite de ciclurile normale de pe magistrala, prin faptul ca sunt indivizibile.
In ceea ce priveste planificarea in sistemul multiprocesor exista 2 posibilitati ca un proces sa fie rulat:
a). Orice proces poate fi rulat pe orice CPU
b). Procesele sunt grupate in submultimi, fiecare din ele alocate unui CPU. Posibilitatile de executie sunt:
- prin alocarea de prioritati proceselor
- prin copii ale situatiei sistemului

2. Sistemele multiprocesor cu memorie locala(proprie)-acestea sunt configurate ca sisteme independente de calcul cu resurse proprii(memorie), unicul mod de folosire in comun a datelor fiind cel de transmisie.

Bineinteles ca timpul de obtinere a datelor, depinde de distanta dintre procesoare si aici un rol importanta il joaca reteaua de comunicatie intre procesoare.

8.4. Organizarea programelor in sisteme multiprocesor

Aceasta este foarte importanta in mentinerea si exploatarea paralelismului sistemelor de calcul.

Un sistem cu arhitectura paralela hardware trebuie exploatat printr-un software gandit paralel, adica algoritmul aplicatiei trebuie gandit paralel.

Organizarea programelor e dependenta de 2 mecanisme fundamentale:

a) mecanismul comenzilor(control mechanism)
b) mecanismul datelor(data mechanism)

si de 3 mari categorii de organizare a programelor:

1) flux de comenzi(control flow)
2) flux de date(data flow)
3) flux de cereri(demand flow)

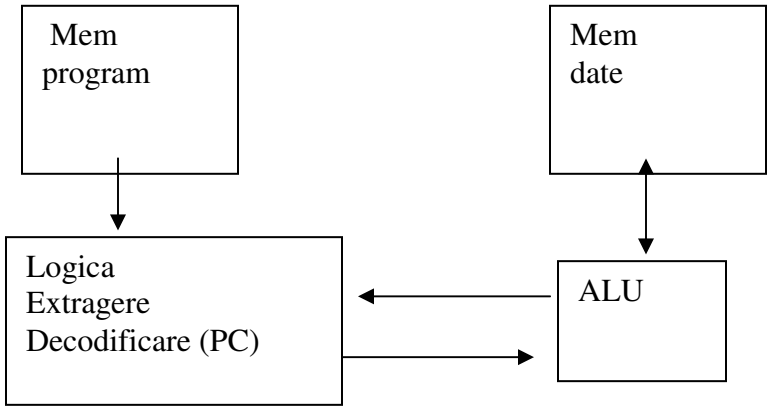
Exista 3 mari categorii de organizare a programelor, in functie de momentul in care se executa o instructiune, acestea fiind:
a1) medii cu derularea evenimentelor impusa de comenzi(command driver enviroment)
a2) medii cu derularea evenimentelor impusa de date(data driven enviroment)
a3) medii cu derularea evenimentelor impusa de cerere(demand driven enviroment)

A) **Mecanismul comenzilor** defineste modul in care o instructiune produce executia alteia si unde la a1) logica de comanda determina ordinea de executie, programele fiind scrise secvential, un contor hardware permitand baleierea secventiala a instructiunilor.
La a2) derularea evenimentelor e impusa de date, o instructiune executandu-se odata ce toti operanzii sai sunt disponibili si rezultatul e solicitat in vederea efetuarii urmatoarei operatii.
B) **Mecanismul datelor** defineste modul in care un operand e transferat instructiunii consumatoare, si aici exista 3 clase:
b1) transfer prin referire (by reference) in care referirea la o unica locatie de memorie e inglobata in fiecare instructiune de acces.
b2) transfer prin valoare (by value) in care instructiunea producatoare transfera celei consumatoare valoarea operandului si nu o referire la el.
b3) transfer prin nume simbolic (by literal) in care operandul e cunoscut in momentul compilarii si cand o copie a sa e inglobata in chiar codul instructiunii.

Fluxul de comenzi- specific calculatoarelor cu arhitectura von Neumann are urmatoarele caracteristici:

- dispune de o memorie globala, adresabila, organizata liniar, in care se pastreaza programele si datele.
- poseda un contor al programului ce pastreaza adresa urmatoarei instructiuni de executat (actualizat implicit sau explicit)
- instructiunile sunt stocate in memoria program, PC parcurgand prin incrementare listele(realizate de programator)
- transferul intre listele producatoare si consumatoare se face prin locatii de memorie utilizate in comun => referire (mecanismul datelor)

- schema bloc:



Flux de comenzi cu trasee multiple- cand un program cu traseu unic de comenzi e desfacut in mai multe procese care coopereaza=> atunci e necesar un mecanism de sincronizare(unii operanzi nu au fost inca evaluati de celelalte procesoare)
-mecanismul comenzii trebuie sa execute instructiunea cand PC-ul o indica si cand operanzii evaluati de alte procesoare sunt disponibili=> trebuie indicat momentul in care se face sincronizarea , apar consumuri suplimentare datorate comunicarii intre procesoare.

Sincronizarea fluxurilor multiple de comenzi
Exista 3 metode

a) folosirea unui compilator cu paralelizare
b) folosirea unor declaratii speciale intr-un limbaj conventional
c) utilizarea unui limbaj functional

a) programul se scrie cu mare atentie asupra datelor si independentelor intr-un limbaj conventional(sa nu se marcheze paralelismul) apoi compilarea acestuia cu un compilator cu paralelizare(exista in acest sens Th Brandes, Somer-Germanis
b) dialecte concurente ale limbajelor secvential
programatorul e cel cel care scrie programul paralel, cunoscand taskurile ce pot fi executate paralel si a punctelor care necesita sincronizarea OCCAM, ADA, CSP, concurent Pascal, paralel C, concurent Fortaan.
-la OCCAM – constructii PAR, SEQ, ALT si primitivele :=, ?, !
- sincronizarea se efectueaza prin deplanificarea proceselor ce comunica si apar replanificate(procesul nu ramane inactiv- executa celelalte procese din lista procesoarelor active.
c) limbajele functionale- ce ofera specificarea explicita a paralelismului, interzicand constructiile cu interdependente obscure

9. SISTEME DE CALCUL ORIENTATE PE FLUX DE DATE (DATA FLOW COMPUTERS)

9.1. Caracteristici generale

Programele in masina data flow se reprezinta prin grafuri orientate indicand interdependenta datelor intre instructiunile producatoare si consumatoare.

Un graf al fluxului de date consta din noduri si ramuri. Un nod reprezinta o interactiune ce se executa atunci cand operanzi devin disponibili.O ramura defineste interdependenta datelor intre 2 noduri. Un nod se poate executa cand exista un operand pe fiecare ramura de intrare.

Cand datelor mai multe actiuni sunt disponibile ele se pot executa concurent daca exista suficiente procesoare disponibile. Cand numai un procesor e disponibil instructiunile se executa secvential, intr-o ordine arbitrara (la momente de timp ce nu pot fi specificate $\Rightarrow$ Masina data flow nu se poate utiliza in aplicatii in care timpul e o resursa critica).

Executia instructiunilor are loc in trei faze:

1)- Se consuma datele existente pe ramurile de intrare
2)- Se efectueaza operatii specificate
3)- Se plaseaza rezultatul pe ramura de iesire

Masina ofera pe langa utilizarea automata a paralelismului, sincronizare automata intre instructiunile productoare si consumatoare.

Transferul se face prin valoare, intre instructiuni transferandu-se date nu referiri la acestea.

Pe ramuri variabilele-sunt de fapt constante, deci unei variabile i se poate atasa o valoare doar o singura data(fiecare ramura e unica).

Dezavantaje:

a) necesar un volum mare de comunicatii intre procesoare.Rezultatul executiei se transmite tuturor instructiunilor ce au nevoie de el, nu numai prin valoare, ci si toate detaliile de indentificare a instructiuniilor de destinatie
b) si un consum sporit de resurse(overhead)

Consumurile suplimentre (Overhead) - se impart in trei categorii:

1- cele aferente determinarii instructiuniilor executabile
2- cele asociate intarzierilor de comunicare
3- cele asociate instructiunilor ce nu efectueaza calcule

1)Pt. determinare instructiuni ce va executa e necesara a tine evidenta operanzilor disponibili.La aparitia unui nou operand se actualizeaza contorul si continutul sau se compara cu numarul necesar de operanzi.

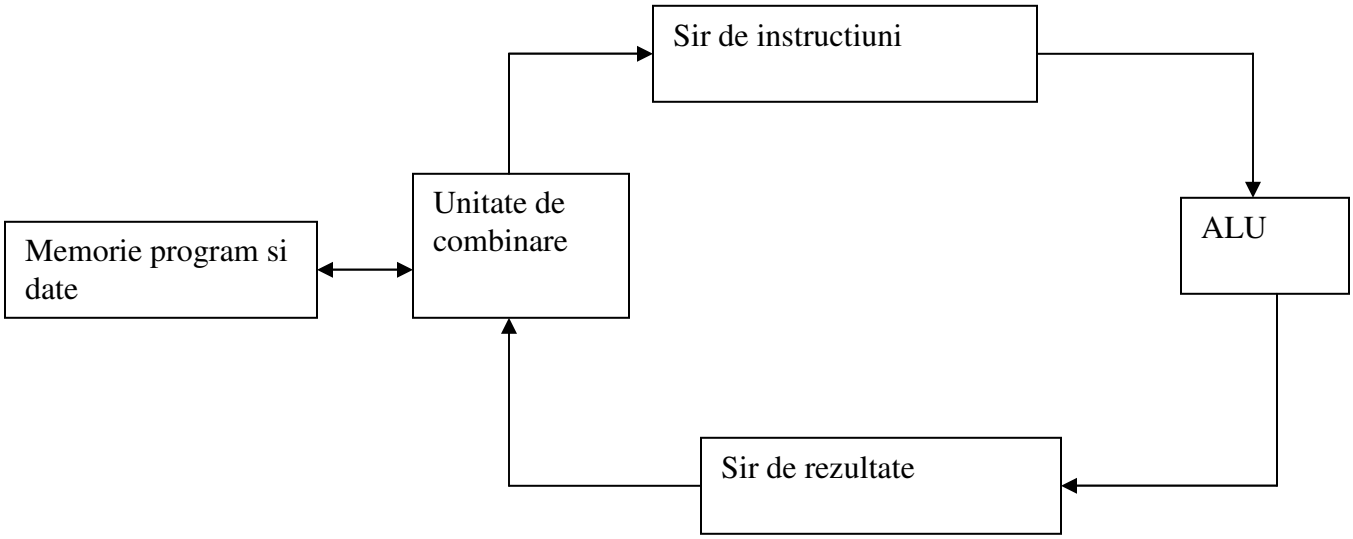
O instructiune ce devine executabila se ataseaza la o lista dinamica de instructiuni executabile, iar cand o resursa devine disponibila trebuie sa se ataseze prima instructiune executabila di lista $\Rightarrow$ sunt mai prioritare instructiunile mai scurte ce se executa mai repede.

2)Intarzierele in procesul de comunicare sunt de doua tipuri:

a) latenta in transmisii a opranzilor de la nodul producator la cel consumator(=timpul de stabilire a traseului intre noduri+timpul de transmisie propriu-zisa)
b) intarzierele in asteptare- provocate de lipsa de canale de comunicatii libere

3)instructiuni ce nu efectueaza calcule(de exemplu cele necesare pt. actualizarea contorului operandului), destul de multe introduc si ele intarzieri.

9.2. Arhitectura calculatorului data-flow



Unitatea de combinare(MU)-rol similar unitati de codificare van Newmann folosit pt. Determinarea urmatoarei instructiuni de executat.

Ea –extrage primul rezultat din sirul de asteptare a rezultatelor

- localizeaza instructiunea de destinatie
- memoreaza rezultatul
- actualizeaza contorul operanzilor disponibili ai instructiunilor
- verifica disponibilitatea tuturor operanzilor si daca sunt toti disponibili plaseaza instructiunea si operanzii sai in sirul de asteptare a instructiunilor.

UAL- extrage cand e libera prima instructiune din sirul de asteptare;o executa si plaseaza rezultatul in sirul de asteptare a rezultatului.

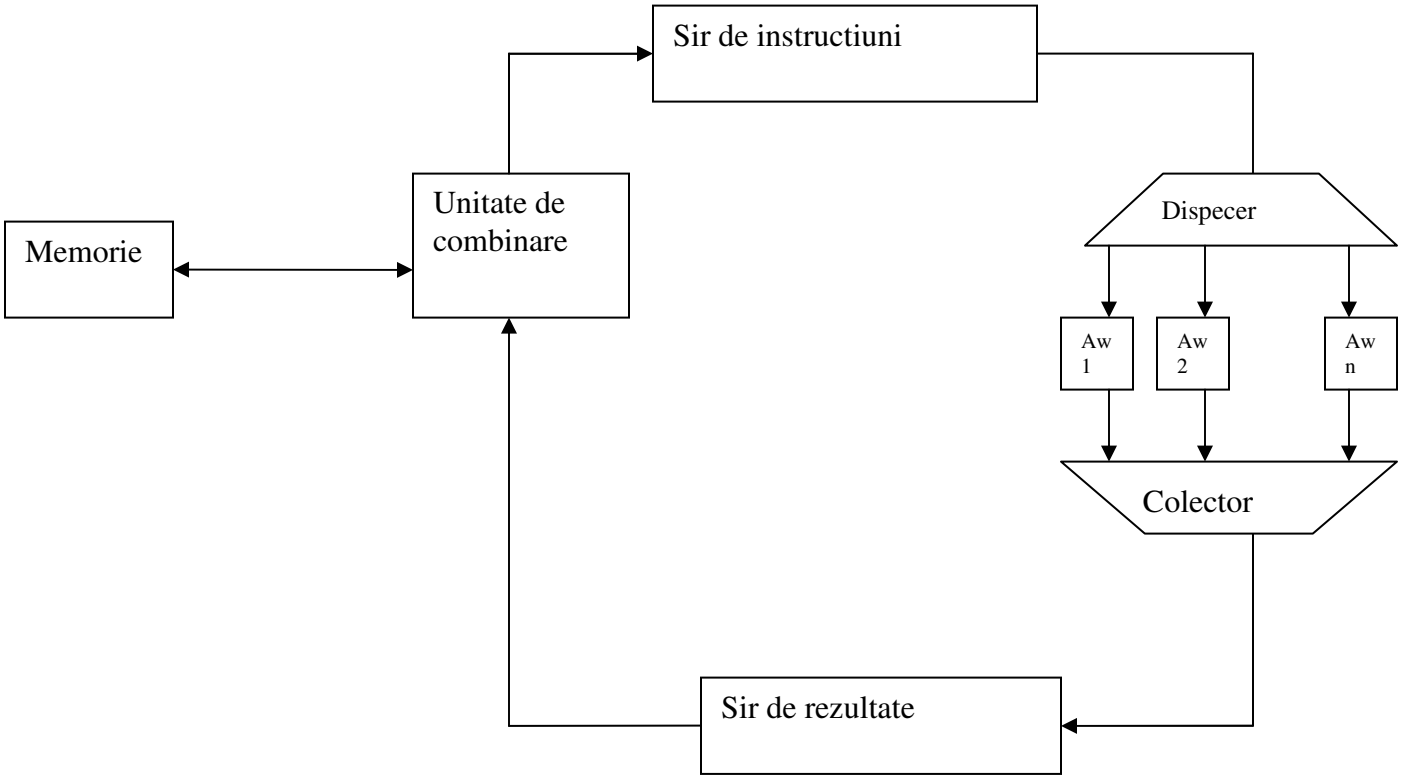
Paralelismul- oferit de operarea concurenta a MU si Aw asupra instructiuni diferite(~ piperline).

Fluxul de date in structurile multiprocesor

Exista doua metode:

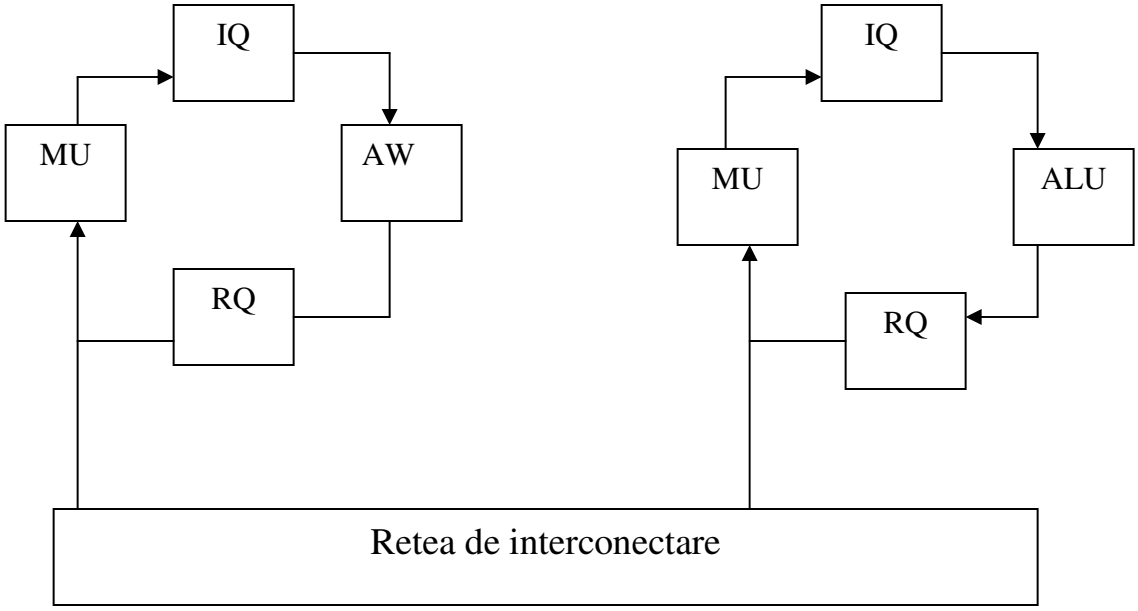
a) Conectarea mai multor ALU la sirul de asteptare a instructiunilor numite calculatoare data flow cu UAL multiple(multi ALU-data flow computer)
b) Interconectarea mai multor sisteme monoprocesor intr-o arhitectura numita calculator data flow multi-inel(multiring- data flow computer)

1)Multi ALU



Dispecerul unic echilibreaza incarcarea frecventei Aw atata timp cat exista suficiente instructiuni executabile(~ 1G Aw pt ca unitatea de combinare sa poata manipula rezultatele).

2)

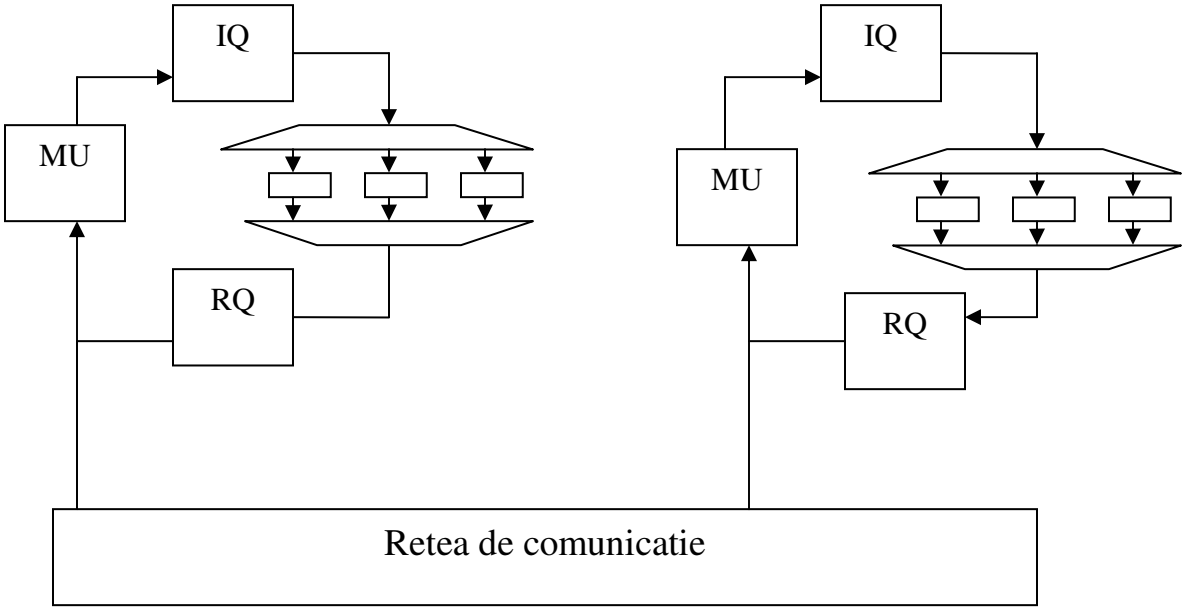


Fiecarui MU i se furnizeaza o instructiune de prelucrat.Exista posibilitatea ca o MU vida iar celelalte prea incarcate.Rezultatele sunt dificil dirijate trebuie recunoscuta MU careia i se atasat instructiunea.

Ramane complexitatea determinarii celui mai scurt traseu sursa-destinatie.

3)Calculatoare data flow multi inel si multi ALU

Numarul de instructiuni atasate fiecarei metode e crescut.



10. REȚELE DE CALCULATOARE. INTERNET

10.1. Internetul

Internetul a devenit una din preocupările centrale în toate domeniile. El este integrat în strategiile de comunicare internă şi externă prin crearea de servere Web si de arhive de date compatibile cu tehnologiile Internet actuale şi nu numai. Internetul este la îndemâna tuturor pentru obținerea de informații sau alte servicii. Nu este necesar decât un calculator şi un modem, iar conectarea la Internet se face imediat. Materialul de față îşi propune o prezentare pe scurt a câtorva cunoştințe fundamentale necesare pentru a înțelege şi utiliza mai eficient acest popular serviciu.

Cuvântul englezesc *Internet* se traduce ca interconexiune de rețele de calculatoare ("net" = rețea de calculatoare). Internetul reprezintă deci o rețea mondială de calculatoare.
O altă denumire pentru Internet o reprezintă *Word Wide Web* (WWW), unde "web" înseamnă pânză de păianjen, deoarece calculatoarele par legate între ele sub această formă.

10.2. Rețele de calculatoare

10.2.1. Definiție

O *rețea de calculatoare* este o colecție de calculatoare autonome interconectate prin canale de comunicație (cabluri, microunde, etc.) prin care acestea interschimbă informație. Între calculatoarele interconectate nu există relații de subordonare.
Avantajele rețelelor sunt multiple:

- Disponibilitatea resurselor în orice punct din rețea. Prin resurse înțelegem programe, date, echipamente.
- Raport preț-performanță foarte bun.
- Fiabilitate mai mare prin posibilitatea de a realiza replici (copii) pe mai multe calculatoare astfel încât la căderea unuia, celelalte copii să fie disponibile.
- Mediu puternic de comunicație deoarece schimbările din rețea sunt sesizabile imediat.
- Sisteme deschise, scalabile în care se pot adăuga noi calculatoare.

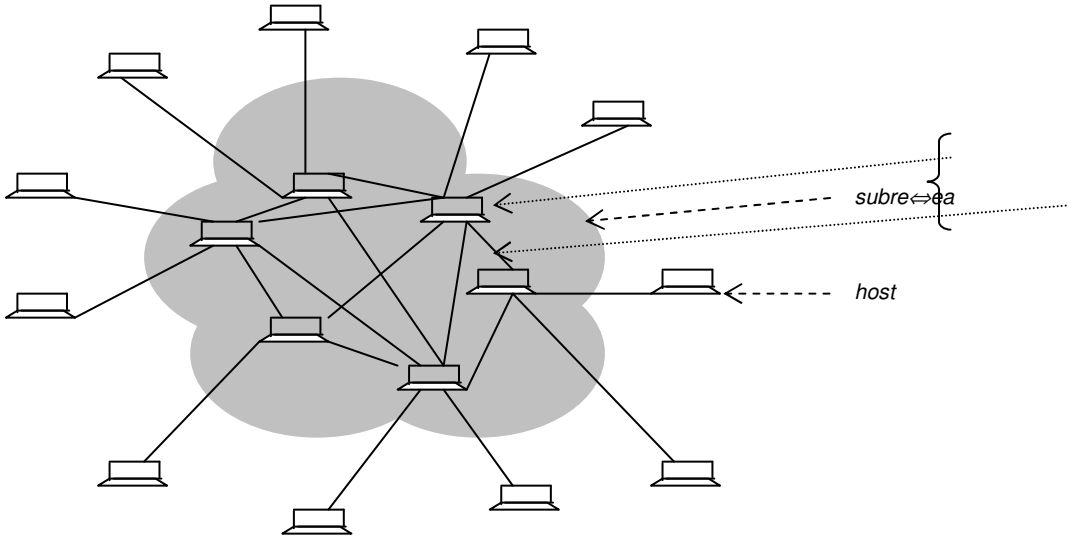
Structura rețelei se referă la caracteristicile tehnice ale rețelei. Intr-o rețea, calculatoarele execută programe utilizator şi comunică mesaje ele. Calculatoarele pe care se execută programele utilizator se numesc "gazdă" (*host*) sau sisteme terminale (*end-system*). Comunicația dintre ele se realizează prin *subrețeaua de comunicație* (*subnet*, pe scurt *net*).

10.2.1 Structura rețelei de calculatoare

- *subrețea* – care are sarcina de a transporta mesaje de la host la host
- *hosturi* (calculatoare gazda) pe care se lanseaza aplicatiile

Subrețeaua este *formată din*

- *noduri sau elemente de comutare* – care sunt calculatoare specializate folosite pentru a conecta *liniile de transmisie*
- *linii de transmisie* care reprezintă canalele de comunicație



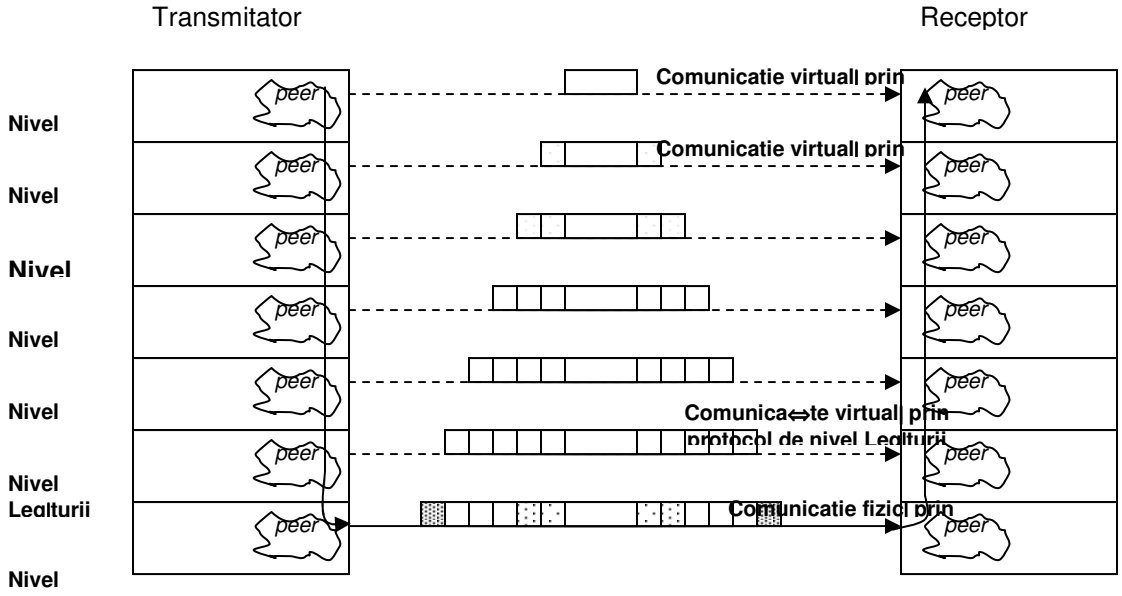
Există două tipuri de subrețele:

- *subrețele punct la punct* - în care pachetele transmise în rețea sunt primite în fiecare nod intermediar, memorate şi retransmise până la destinație
- *subrețele broadcast* - în care există un singur canal de comunicație partajat de toate calculatoarele din rețea. Calculatoarele care nu sunt destinatarele pachetului îl ignoră.

10.2.2. Arhitectura rețelei de calculatoare

Rețelele de calculatoare sunt proiectate într-un mod foarte bine structurat. Pentru a se reduce complexitatea în proiectare, dar şi pentru uşurință în întreținere şi modificare, rețelele sunt organizate într-o serie de niveluri ierarhice. Scopul fiecărui nivel este de a oferi servicii nivelului superior, ascunzând față de acesta detaliile referitoare la modul de implementare a serviciilor.
Nivelul *n* de pe o maşină comunică cu nivelul *n* de pe altă maşină printr-un *protocol*. Entitățile care comunică se numesc *entități pereche* (peer). Comunicația astfel realizată este *virtuală*.
In realitate datele călătoresc prin mediul fizic de comunicație care reprezintă cel mai de jos nivel al ierarhiei.
Pentru rețelele de calculatoare ISO (International Standard Organization) a elaborat standardul *OSI (Open System Interconnection*). Corespunzător acestui standard, arhitectura rețelelor de calculatoare cuprinde şapte niveluri: Aplicație, Prezentare, Sesiune, Transport, Rețea, Legăturii de date, Fizic.
De obicei utilizatorul lucrează la cel mai superior nivel (Aplicație) al rețelei. Mesajul este transmis de la nivelul Aplicație al calculatorului transmițător la nivelul Aplicație al calculatorului receptor respectând regulile protocolului de nivel. Pentru transmiterea mesajului însă, nivelul Aplicație transmițător utilizează serviciile oferite de nivelul adiacent inferior, nivelul Prezentare după standardul OSI, iar mesajul de transmis este împachetat cu informație de control pentru transmisia virtuală de la acest nivel. Acesta la rândul său utilizeaza serviciile oferite de nivelul inferior care împachetează mesajul cu informația de control necesară protocolului de nivel. In cele din urmă mesajul ajunge la cel mai inferior nivel, cel Fizic, la care se realizează comunicația fizică. Informația primită este preluată de nivelul receptor imediat superior celui Fizic (de obicei cel al Legăturii de

date) despachetat de informația de control şi transmis nivelului superior, până ce mesajul ajunge în forma în care a fost transmis la nivelul Aplicație receptor.
Dirijarea pachetelor în subrețea este realizată de nivelul Rețea, iar cominicația end-to-end (adică de la host la host) este realizată de nivelul Transport. Rețelele existente anterior standardului OSI evident că nu-l respectă, de exemplu rețeaua ARPANET (prima rețea realizată). La aceasta nivelul Rețea şi nivelul Transport sunt unite. Pentru ele exista un singur protocol TCP-IP. TCP (Transmision Control Protocol) este un protocol pentru nivelul Transport, iar IP (Internet Protocol) pentru nivelul Rețea în standardul OSI.



Dintre serviciile oferite de rețelele de calculatoare sunt:
- transferul de fişiere (FTP – File Transfer Protocol)
- partajarea fişierelor
- poşta electronică (e-mail)
- emulare de terminal (Telnet)
- acces la informații stocate în bazele de date aflate la distanță
- comenzi la distanta (RPC – Remote Procedure Call)
- gestiune (administrarea retelei)

10.2.3. Arhitectura aplicațiilor Internet

Aplicațiile Internet se execută la nivelul Aplicație al rețelelor de calculatoare. Arhitectura utilizată pentru aplicațiile Internet este arhitectura client-server.

ARHITECTURA CLIENT-SERVER
În aceasta arhitectura pe anumite calculatoare cu resurse mai mari ruleaza, în background, procese de tip *server* (procese *daemon*). Procesele *client* care ruleza în general pe alte calculatoare (se pot executa si pe acelasi calculator) fac *cereri*, printr-un anumit protocol, catre servere. Serverele *raspund* la cererile clientilor. Procesele client recepționează datele, le interpretează si afişează rezultatele. Arhitectura se mai numeşte *cerere-răspuns.*

Pentru Internet aplicația tipică de afişare de informații este cea în care:

- clientul = Browser
- serverul = server de Web

Cele mai populare browsere sunt: Netscape şi Internet Explorer.
Cele mai populare servere Web: Apache, Netscape, WinNT Web Server, etc.

Protocolul cel mai utilizat de browsere pentru obtinerea informatiilor este HTTP (HyperText Transfer Protocol).
Pentru scrierea paginilor de Web există mai multe tehnologii, dar cea mai utilizată este HTML (HyperText Markup Language)
HTML este un subset al limbajului mai complex SGML(Standard Generalized Markup Language).

Modelele de baza pentru arhitecturile client-server sunt pe două nivele (two-tier) sau trei nivele (three-tier) după modul în care logica aplicației este distribuită între client şi server.
În această arhitectură sistemul este divizat în două părți:

- o parte client (front-end)
- o parte server (back-end)

Partea server execută sofware-ul de server şi manipulează funcțiile cerute pentru accesul concurent şi partajat la date. Partea server primeşte şi procesează comenzi de la partea client a aplicației.

Partea client reprezintă partea front-end a aplicatiei, cuprinzand interfața cu utilizatorul. Ea interacționeaza cu utilizatorul prin intermediul tastaturii, display-ului, mouse-ului. Partea client nu prezinta responsabilități referitoare la accesul datelor, concentrându-se pe cererile către server şi procesarea şi prezentarea datelor la/către acesta.

O arhitectură mai flexibilă pentru aplicațiile Internet o reprezintă arhitectura pe mai multe nivele: m*ulti-tier*.
In arhitectura multi-tier avem urmatoarele componente:

- un client sau inițiator de proces care porneşte operația
- unul sau mai multe servere de aplicație care realizează parți ale operației. Un server de aplicație este un proces furnizează accesul la date pentru client şi realizează parți din procesarea interogarilor, ducând astfel la eliberarea încarcarii serverului (de daze de date de obicei). El serveşte ca interfață între client şi serverele de baze de date şi mai furnizează şi un nivel adițional de securitate. Server-ul de Web dintr-o aplicație Internet este un server de aplicație.
- serverul de baze de date se afla la celălalt capăt şi serveşte ca repository pentru cele mai multe date utilizate în operații.

Aceasta arhitectură permite ca serverul de aplicație să valideze datele clientului, să realizeze conexiunea la baza de date şi să obtină datele cerute de client.

10.2.4. Identificarea resurselor

Identificarea calculatoarelor

- prin adresa de IP (Internet Protocol) formată din patru grupe de cifre care identifică în mod unic calculatorul. Poate fi specificata manual sau automat generata de DHCP.
- prin nume logic

Numele logic este format din trei nivele:

- nivelul de domeniu: 7 domenii internationale (.com, .int, .net, .org, .edu, .gov, .mil) si domenii nationale (.ro, uk, .us, fr...). Indică organizația.
- nivelul de subdomeniu – pentru întreprinderile mari sau furnizorii de servicii Internet (univ-ovidius, microsoft, imb, ...) care indică serverul central.
- nivelul de serviciu oferit: www (server web), ftp (transfer de fisiere), etc.

Corespondenta între numele logice şi adresele numerice este realizata de DNS, serverul de nume de domeniu.

Identificarea resurselor

Identificarea resurselor se face prin URI (Universal Resouce Identifier). Acesta poate fi :

- o locație identificata prin URL (Unifversal Resource Locator)
- un nume de resursa – identificat prin URN (Universal Resource Name)

Acesta contine protocolul utilizat pentru comunicare, serverul accesat, localizarea resursei pe calculatorul care gazduieşte procesul server:

<protocol>:<nume_logic>[:port]/cale/resursa

De exemplu pentru afisarea paginii Web a universitatii “Ovidius”:

http://www.univ-ovidius.ro

Pentru protocolul HTTP adresa implicita portului la care serverul de web asculta (asteapta cererile) este 80.

Sumar:

Internetul este o rețea mondială de rețele de calculatoare care oferă servicii utilzatorilor săi.

O rețea de calculatoare este o colecție de calculatoare interconecate prin canale de comunicație. Ea este alcătuită din subrețea (noduri, linii de transmisie) şi calculatoare host. Arhitectura rețelelor de calculatoare este multinivel (standardul OSI). Comunicația între nivelurile corespondente este virtuală şi se realizează după un protocol de nivel.

Aplicațiile Internet se execută la nivelul Aplicație şi au arhitectură client-server (cerere-răspuns).

Identificarea calculatoarelor se face prin adresă de IP sau nume logic, iar a resurselor prin URI (URL sau URN).

11. CONCEPTE UTILIZATE ÎN SISTEMELE MODERNE DE CALCUL

11.1. Contextul actual al sistemelor de calcul

1. Cel mai important: Progresul tehnologic ce include tehnologia de integrare si crestere a vitezei ceasului (cresterea resurselor).
   Legea hardware-ului. Directii de dezvoltare: CISC, RISC, JVM, calculatoare paralele.
2. Dezvoltarea tehnicilor moderne de proiectare si implementare (cresterea performantei). Legea software-ului.
3. Existenta standardelor si sistemelor gata facute (simplitate).
4. Pret scazut (disponibilitate)
5. Internetul (deschidere)

Rezultat
1. Posibilitatea de a realiza sisteme complexe, chiar de timp real, utilizand tehnici nepretentioase.
2. Granita dintre hardware si software nedefinita. Tehnicile descoperite intr-un domeniu sunt valabile si in celalalt. Teoria este generala.
3. Accent pus pe problematica generala a sistemului. Se poate privi de la un nivel superior fara a fi necesare detalii.
4. Utilizarea componentelor, produse gata facute pentru constructia de sisteme
5. Cresterea posibiltatilor de comunicare

Probleme
1. Progesul rapid => schimbarea tehnologiilor, componentelor => dificultati in constructie (timp scurt)
2. Alegerea arhitecturii generale a sistemului din componente. Problema combinarii lor. Lipsa de flexibilitate (greu de implementat functii specifice).
3. Necesitati sporite de comunicare. Overhead. Scalabilitate.
4. Sisteme prea complexe, prezentate prea general, sinteza mare=>greu de urmarit

Internetul, in ziua de astazi, reprezinta o provocare pentru toate tipurile de sisteme de calcul. Conectarea sistemelor de conducere la Internet poate sa aduca un numar de avantaje ca: accesul la site-urile Web, monitorizarea la distanta a obiectului controlat, diagnoza la distanta a caderilor sistem, etc.
Exista insa doua mari probleme legate de tehnologia Internet in aplicatiile de timp real: lipsa de securitate si performanta temporala impredictibila. Daca securitatea este aproape rezolvata avand chiar comertul electronic care se face pe Internet, cea dea doua ramane si este greu sa se garanteze performanta pentru aplicatiile de timp real.

11.2. Arhitectura sistemelor informatice

O problema critica in proiectarea si constructia oricarui sistem informatic complex este arhitectura lui: **organizarea ca o colectie de componente care interactioneaza**.

Arhitectura sistemului se situeaza intre cerinte sistemului si implementarea acestuia.

Arhitectura **furnizeaza o descriere abstracta a unui sistem**, in timp ce expune anumite proprietati, ascunde altele. Aceasta reprezentare furnizeza o **viziune globala** a sistemului, permitand proiectantilor sa rationeze referitor la abilitatea sistemului de a satisface anumite cerinte si sa sugereze o schita pentru constructia si compozitia sistemului.

Arhitectura sistem = dispunerea si interconectarea componentelor pentru a obtine functionalitatea dorita a sistemului.

11.2.1. Arhitecturi generale

Arhitectura multistrat – niveluri ierarhice. Un nivel inferior ofera suport nivelului superior pentru executia functiilor sale
Decompozitia functionala – descompunere a componentelor dupa functiile realizate
Decompozitia conceptuala – descompunere a sistemului dupa entitatile identificate (ce inglobeaza toata functionalitatea obiectului).
Decompoziția spațială – utilizată în sistemele distribuite pentru ca resursele de calcul să fie mai aproape de locul de utilizare
Decompoziția temporală – utilizată în sistemele de timp-real pentru sincronizarea operațiilor

Arhitectura software a unui sistem are următoarele caracteristici:
• ghidează procesul de partiționare a funcțiilor unui sistem
• specifică împărțirea sistemului în nivele
• specifică interfața între nivele
• trebuie să ia în considerația utilizabilitatea şi problematica sistemului.

11.2.2. Arhitectura multinivel

Arhitectura multi strat este arhitectura care oferă cea mai mare modularitate sistemelor şi de aici flexibilitate, uşurință în întreținere şi modificare.
În arhitectura multinivel componentele sistemului sunt dispuse pe verticală. Principiul pe care sunt dispuse nivelurile este următorul: nivelul inferior oferă servicii nivelului superior ca o cutie neagra fără a da detalii de implementare a functionalitatii sale.
Sisteme sunt bine structurate, cu flexibilitate oferită de numărul de niveluri.
Nivele diferite de abstractizare ale aceluiaşi sistem de la modelul fizic al datelor, la modelul logic al datelor, rețele, specificații de aplicație, modelul business al aplicației.
Dezavantajul major pentru sistemele cu mai multe niveluri îl reprezintă consumul suplimentar de resurse (overhead) introdus de fiecare nivel.
Există, corespunzător, mai multe **nivele de granularitate**. La nivelul inferior granularitatea este mai fină, implementându-se module cu coeziune mare (în general o funcție pentru un modul). La nivel superior granularitatea este mare, implementându-se sisteme integrate cu mai multe funcțiuni.

Exemple:
Arhitectura ierarhică a sistemului de calcul. Fiecare nivel reprezinta o masina virtuala programata in limbajul masina al nivelului respectiv. Executie prin translatare sau interpretare.
Modelul OSI pentru rețele de calculatoare

11.2.3. Arhitectura client – server

Ideea – de a structura sistemele ca un grup de componente care cooperează numite servere şi care oferă servicii utilizatorilor numiți clienți.
Modelul client server se bazează pe un protocol simplu cerere/răspuns (request/reply).
Avantaje:
Simplitate. Nu este necesară o conexiune anterioară cererii, iar răspunsul vine ca o confirmare (acknowledgement) la cerere.
Eficiență – prin consistența automată a resurselor
In consecință arhitectura are doua parti:

- parte server reprezentata printr-o componentă server care are rol de a servi componentele client ca raspuns la cererile acestora
- parte client reprezentata prin procese client care fac cereri catre server. Aceasta parte interactioneaza cu utilizatorul si nu are responsabilitati de acces conectrandu-se pe cererea, procesarea si prezentarea datelor.

Arhitectura client-server se mai numeşte arhitectură cerere-răspuns.
Arhitectura client-server este o arhitectură centralizată în care problemele de consistență a datelor sunt direct rezolvate. Pentru arhitectura client-server, există realizate sisteme care gestionează resursele de calcul şi oferă servicii clienților. Exemple tipice de utilizare a acestei arhitecturi sunt în următoarele sisteme:

- sisteme de gestiune a bazelor de date
- aplicații Internet.

11.2.4. Arhitectura multi-tier

Arhitectura multi-tier este o arhitectura client/server cu mai multe niveluri. În acestă arhitectură componenta server este la rândul ei client pentru o altă componentă server.

Arhitectura Middleware

Arhitectura Middleware reprezintă o soluție software pe un nivel intermediar care oferă servicii pentru dezvoltarea aplicațiilor distribuite.
Arhitectura Middleware simplifica constructia sistemelor distribuite:

- rezolva problemele de eterogenitate
- faciliteaza comunicatia si coordonarea componentelor distribuite
- solutioneaza probleme de toleranta la defecte
- rezolva probleme de securitate.

Exemple de soluții Middleware: CORBA, DCOM, J2EE.

11.2.5. Arhitectura orientată obiect

Componentele unui sistem construit intr-o arhitectura orientata obiect sunt obiectele. Un obiect se caracterizeaza prin faptul ca este responsabil cu mentinerea integritatii reprezentarii informatiilor pe care le incapsuleaza. Obiectele unui sistem sunt modelate utilizand tipuri abstracte de date.

Obiectul

Conceptul de obiect este foarte utilizat in dezvoltatrea sistemelor de calcul actuale.
pentru a modela obiectele lumii reale.

Obiectele *colaborează* intre ele pentru obtinerea funcționalității dorite.

Modelul orientat obiect

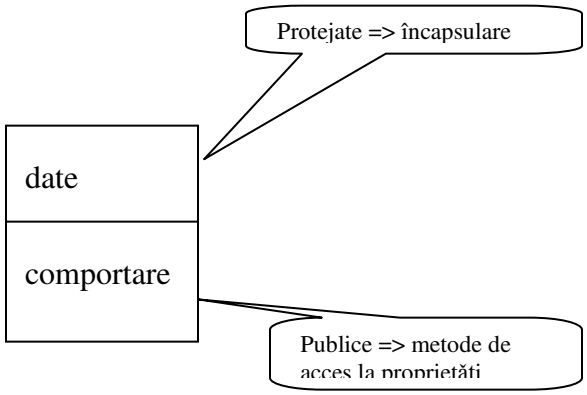
Un *model* este o descriere formală a unui aspect cheie al unui sistem dintr-un anumit punct de vedere. El reprezintă întotdeauna o *abstractizare a lumii reale*, ignorând anumite aspecte care nu interesează pe utilizatorul de model.

Modelele structurate separă datele de funcții. In proiectarea structurata modulele program erau divizate a.i. sa indeplineaca doua criterii importante:

- coeziune mare a unui modul = daca se poate sa implementeze o singura functie
- cuplare mica intre module = cat mai putine interdependente a.i. modificarea unui modul sa nu le inflenteze pe celelalte.

Modelele orientate obiect unifică datele şi funcţiile care operează asupra lor în componente software numite *obiecte*. Consecința o reprezintă o scădere a coeziunii.

Clasele reprezintă *abstractizari ale obiectelor*. Ele capturează obiectele de acelaşi fel: cu aceeaşi structură de date şi aceeaşi comportare.
În acest context *obiectele* reprezintă *instanțe ale claselor.*



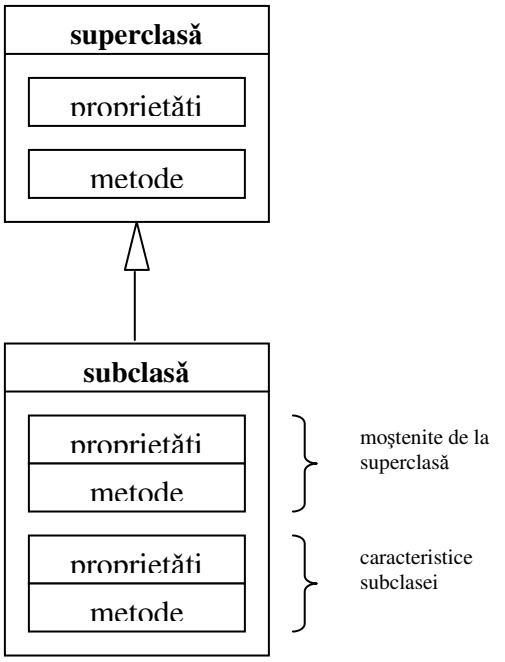
Datele sunt numite *proprietăți* în programarea OO şi *atribute* în proiectarea OO.
Comportarea este implementată în funcții care acționează asupra datelor, numite *metode* în programarea OO şi *operații* în proiectarea OO.
Proiectarea clasei se referă la specificarea datelor şi a comportamentului.
Principiile de bază ale modelului orientat obiect sunt:

- încapsularea
- moştenirea
- polimorfismul

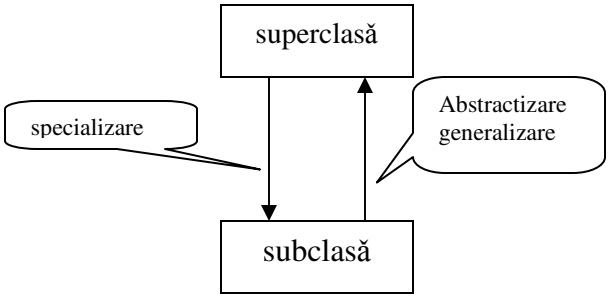
Incapsularea obiectelor se referă la proprietățile clasei care pot fi accesate numai prin metodele ei. Avantajele obținute sunt:

- siguranța – utilizatorii nu pot afecta starea obiectului decât prin metode
- fiabilitatea – dezvoltatorii pot realiza modificări asupra obiectului fără ai afecta pe utilizatori.

Moştenirea reprezintă una din cheile arhitecturilor moderne şi se referă la partajarea definițiilor şi regulilor. O sublasă moşteneşte toate prorietățile şi metodele superclase. Deci o subclasă poate utiliza o proprietate care este specificată de o superclasă.



Conceptual mostenirea poate fi reprezentată astfel:



Polimorfismul se referă la comportament. Aceeaşi metodă cu implementări diferite. Limbajul Java, implementeaza doua tipuri de polimorfism:

- *static* = *overloading* - se referă la metodele cu acelaşi nume, dar parametrii diferiți (semnatura metodei). Identificarea metodei este făcută la compilare după parametrii definiți pentru metoda respectivă.
- *Dinamic* = *overriding* - apare în ierarhiile de clase şi se referă la metode cu acelaşi nume şi semnătură într-o ierarhie de clase. Identificarea metodei făcându-se la execuție.

Interfețe – specifică un cadru de comunicare între obiecte diferite. Fiecare clasă care utilizează interfața trebuie să furnizeze o implementare particulară a interfeței.
În programarea OO ea este specificată prin *metode abstracte.*

Evenimente

Interfetele aplicatiilor sunt grafice, i.e. utilizatorul interactioneaza cu aplicatia prin utilizarea obiectelor grafice. Specificarea interactiunii se realizeaza prin intermediul evenimentelor.

Evenimentele sunt declansate de actiunea tastelor sau butoanelor mouse-ului. Acestea pot fi: Click, Dubluclick, Enter, Focus, etc..

Aplicatia trebuie sa fie pregatita pentru a le trata. Pentru acest lucru o metoda *action_listener*() este atasata fiecarui eveniment.

Aplicatiile moderne au interfata grafica (GUI) si sunt compuse din mai multe obiecte care colaboreaza si care lucreaza intr-un context special specificat de un obiect de tip *container*. Containerul este deasemenea un obiect care ofera suportul necesar comunicarii intre obiecte. Exemple: fereastra, frame, canvas.

Modelul bazat pe componente

Model nou utilizat in realizarea sistemelor moderne de calcul. Reprezinta "industrializarea" dezvoltarii software a sistemelor de calcul bazandu-se pe asamblarea elementelor comune testate in realizarea aplicatiilor. Ceva asemanator cu constructia calculatoarelor de tip PC in ziua de astazi.
Software are insa in plus fata de hardware si dimensiunea abstractizarii. O componenta software nu este limitata de caracteristicile fizice ale obiectului pe care il reprezinta. O componenta software poate asamblata si reasamblata in diferite aplicatii schimbaindu-i-se caracteristicile.
Exista trei nivele de componentizare, numite si granularitati.

- granularitate fina: *controle ActiveX, JavaBean*
- granularitate medie: componente *business* (de aplicatie) care incapsuleaza reguli ale aplicatiei
- granularitate mare, cu cea mai mare rata de reutilizabilitate: *cadre de aplicatie* (*application frameworks*) care furnizeaza infrastructura software pe care toate componentele o folosesc in executia aplicatiei.

Componentele ActiveX (tehnologie Microsoft) si JavaBean (tehnologie Sun)
Un control ActiveX este un obiect *COM (Component Object Model)* care se supune unor reguli in interfetele pe care le suporta.
COM este tehnologia Microsoft pentru realizarea aplicatiilor distribuite, care faciliteaza integrarea aplicatiilor Web si client-server intr-o singura arhitectura unificata. Utilizand COM se pot realiza componente distribuite scrise in limbaje diferite care pot interactiona in retea.
COM+ - ofera facilitati sporite tehnologiei COM:

- servicii integrate care include tranzactii, securitate, acces la bazele de date, cozi de mesaje
- instrumente de programe pentru diferite limbaje
- o biblioteca bogata pentru configurarea aplicatiilor si reutilizarea componentelor.
- Suport pentru interoperare intre utilizatori si dezvoltatori.

CORBA (Common Object Request Broker Architecture) – dezvoltata de OMG (Object Management Group) faciliteaza invocarea de metode asupra obiectelor distribuite in retea. O implementare CORBA implica un *ORB (Object Request Broker)* localizat si la client si la server pentru a crea si gestiona comunicatiile client-server dintre obiecte. Protocolul utilizat pentru comunicare este *IIOP (Internet Inter-ORB Protocol)*.
JavaBeans – arhitectura independenta de platforma (bytecode) pentru dezvoltarea de aplicatii in retea.

O *JavaBean* este un obiect Java care implementeaza interfete standard specifice.
O componenta este un pachet (nu in sensul de pachet software) de software care:

- se executa intr-un container (ca VB sau browser), deci nu este independent
- furnizeaza o interfata utilizator vizuala (grafica)

- este distribuita in forma executabila
- poate expune un grup de proprietati si metode in containerul respectiv
- poate transmite evenimente containerului.

Un **control ActiveX** corespunde descrierii anterioare**.**
Fara interfata utilizator se numeste **componenta ActiveX** sau **obiect COM**.
Componentele care se executa la server (nu utilizeaza interfete) se numesc componente ActiveX.
In terminologia Sun, componentele Desktop se numesc **JavaBeans**, iar cele de la server se numesc **Enterprise JavaBeans**.

Meta – modele
Metamodelele sunt modele pentru descrierea modelelor.
Meta-metamodelele sunt modele pentru descrierea metamodelelor.

Aplicațiile sunt atăt de mari încât nu se mai realizează ca sisteme de sine stătătoare. Ele sunt realizate din componente existente şi utilizează servicii comune disponibile în contextul de implementare destinație.
In timp ce rețelele, în special Internetul, facilitează interoperatibilitatea, aplicațiile din ziua de astăzi necesită interoperarea cu alte aplicații şi partajarea informațiilor între ele.

Prin modelare arhitecții pot administra mai bine complexitatea sistemelor software.
Modelele permit arhitecților separarea proiectării (design) de implementare.

UML (Unified Modelling Language)
XML (eXtensible Markup Language)
MOF (Meta Object Facility) – este un meta-metamodel utilizat pentru descrierea arhitecturii de dezvoltare a software-ului (inclusiv a lui). El descrie:
- limbaje de modelare ca UML
- un set de metamodele de tehnologie cum ar fi: CCM (CORBA Component Model), EJB (Enterprise JavaBeans)
- alte metamodele definite de utilizator

Pentru descrierea unui sistem software, arhitecții software utilizează un limbaj de modelare ca UML pentru a descrie “proiectul”.
Proiectul poate fi apoi îmbunătățit utilizănd modelele de tehnologie şi alte modele pentru a descrie semantica de implementare/execuție (runtime).
Regulile MOF sunt aplicate modelului pentru a defini semantica de interoperatibilitate şi formatul pentru interschimburi.
Această arhitectură permite arhitecților o descriere completă a sistemului.

LABORATOARE

LABORATOR 1

I. Baze de numerație : binar, octal, hexazecimal. Conversii.

1. **Binar** – 2 cifre: 0, 1

2. **Octal** – 8 cifre (0-7), fiecare cifra se scrie pe 3 biti

Binar	Octal
000	- 0
001	- 1
010	- 2
011	- 3
100	- 4
101	- 5
110	– 6
111	- 7

3. **Hexazecimal** – 16 cifre (0-9, a, b, c, d, e, f), fiecare cifra se scrie pe 4 biti

Binar	Hexazecimal
0000	- 0
0001	- 1
0010	- 2
0011	- 3
0100	- 4
0101	- 5
0110	- 6
0111	- 7
1000	- 8
1001	- 9
1010	- A
1011	- B
1100	- C
1101	- D
1110	- E
1111	- F

CONVERSII

1. ZECIMAL -> BINAR

$N = \sum_{i=1}^{n-1} C_i * 2^i$, unde C_i = cifră binară

Metodă. Se identifică coeficienții $C_i = 1$ din dezvoltarea binară.

Algoritm:

a) Se caută cea mai mare putere a lui 2 mai mică decât numărul – reprezintă puterea lui 2 a cărei coeficient este 1
b) Se scade valoarea gasita
c) Dacă valoarea rămasă este mai mică decât 2, STOP
d) Cu valoarea rămasă, salt la pasul a)

Exerciții: Gasiți valoarea binară a numerelor zecimale: 126, 215, 47, 23

2. BINAR -> ZECIMAL

Se calculează valoare zecimală utilizând formula de mai sus.
Exercițiu: Conversii binar->zecimal pentru următoarele numere:
1011101
0110001
10001101

3. BINAR - > HEXAZECIMAL

Se grupează biții de la dreapta spre stânga câte 4. Fiecare grupare reprezintă o cifră hexazecimală.
Exerciții: Exprimați în hexazecimal următoarele numere binare
1000010111011101
1111111000000
1001011010111

4. BINAR -> OCTAL

Se grupează biții de la dreapta spre stânga câte 3. Fiecare grupare reprezintă o cifră octală.
Exerciții: Exprimați în octal numerele binare de mai sus.

5. ZECIMAL -> OCTAL

Se converteşte în binar şi se grupează biții de la dreapta spre stânga câte 3. Fiecare grupare reprezintă o cifră octală.
Exerciții: Gasiți valoarea octală pentru numerele de mai sus.

6. ZECIMAL -> HEXAZECIMAL

Se converteşte în binar şi se grupează biții de la dreapta spre stânga câte 4. Fiecare grupare reprezintă o cifră hexazecimală.
Exerciții: Gasiți valoarea hexazecimală pentru numerele de mai sus.

7. HEXAZECIMAL -> BINAR

Fiecare cifră hexazecimala se exprimă binar.
Exercițiu:

8. HEXAZECIMAL -> OCTAL
9. HEXAZECIMAL -> ZECIMAL
10. OCTAL -> BINAR
11. OCTAL -> HEXAZECIMAL
12. OCTAL -> ZECIMAL

II. Hardware.

Identificati **principalele componente ale calculatorului** la care lucrati si caracteristicile lor:

- Microprocesor
- Memorie RAM
- Memorie externa
  - Discuri logice
    - capacitate
    - spatiu liber pe disc
- Subsistem de I/E
- – identificati driverele instalate pentru urmatoarele dispozitive si modaliatile de configurare ale acestora:
  - Tastatura
  - Mouse
  - Monitor
  - Imprimanta
  - Placa retea
  - Placa sunet

Atentie, se vor studia toate facilitatile oferite de interfata de configurare a dispozitivelor! Retineti setarile initiale si refaceti-le dupa ce ati terminat studiul.

III. Software. Sistemul de operare.

Interfata cu utilizatorul. Moduri de lucru (mod comanda, GUI)

Modul comanda

Identificati modalitatea de lansare a ferestrei de comenzi. In fereastra de comenzi identificati prompterul sistem. Lansati comanda “exit” de parasire a ferestrei.

Modul de lucru GUI

Identificati obiectele afisate in Desktop. Afisati functionalitatea expusa de acestea (click dreapta mouse pe obiectul selectat).
Identificati principalele elemente ale ferestrelor de aplicatii: bara de titlu, meniu, bare de instrumente, etc. Pentru aceasta lansati in executie diferite tipuri de aplicatii: My Computer, Windows Explorer, etc.
Pentru fiecare dintre acestea identificati obiectele prelucrate si functionalitatea expusa de acestea. Ce observati?

Meniul principal – START

Identificati si studiati principalele optiuni ale acestuia: Help, Run, Search, Settings Documents, Programs.

LABORATOR 2

I. Organizarea informatiei pe disc (arborescent). Operatii de gestiune a informatiei stocate.

Aplicatii de gestiune a informatiei pe disc:
My Computer si Windows Explorer.
Studiati optiunile Edit si Tools.

Utilitarele My Computer şi Windows Explorer

1. Studiați funcțiile oferite de cele două aplicații. Atenție – parcurgeți fiecare opțiune în parte. Marcați diferențele dintre cele două aplicații în caiet.
2. Creați următoarea structură arborescentă în folderul curent al grupei dvs.:

```
CAT1    ----CAT11    -----CAT111
        ----CAT12
        ----CAT13    -----CAT131
                     -----CAT132
CAT2    ----CAT21
        ----CAT22
```

3. Creați în folderul CAT111 fişierul fis1.txt, utilizănd Notepad
4. Creați în folderul CAT131 fişierul “fis2.bmp”, utilizănd Paint
5. Setați mediul să ascundă extensia fişierelor
6. Copiați fişierele “fis1” şi “fis2” în CAT2.
7. Redenumiți “fis1” şi fis2 cu “copie1” şi “copie2”, păstrând extensia.
8. Protejați la scriere “copie1”.
9. Setati atributul Hidden lui “copie2”.
10. Setati mediul să nu vizualizeze fisierele marcate “hidden”.
11. Modificați “copie1” şi “copie2”
12. Stergeti “copie2”.
13. Mutati “copie1” in CAT31.
14. Copiati cursul Birotică în CAT22.
15. Afişați informația din folderul “Concepte” in ordine alfabetica, apoi cronologica.
16. Creați catalogul CAT în folderul grupei dvs.
17. Mutați toată structura creată (CAT1 şi CAT2) în CAT.
18. Partajați CAT cu opțiunea de vizualizare, fără modificare, numai pentru utilizatorii din laborator.
19. Vizualizați CAT de pe celelalte calculatoare din laborator.
20. Creați un shortcut pentru CAT In Desktop.
21. Stergeti structura creată.
22. Refaceți structura ştearsă.
23. Stergeti definitiv structura.
24. Ştergeți shortcut-ul creat.

II. Editorul WORD

EDITARE TEXTE

1. Studiați meniul principal al aplicației.
2. Învățați să gestionați barele de instrumente (Toolbars).
   Opțiunea View -> Toolbars – pentru vizualizarea barelor de instrumente. Identificați barele de instrumente.
   Opțiunea View -> Toolbars -> Customize pentru configurarea barelor de instrumente
   - se selectează tabul “Commands” şi prin Drag and Drop (mutare cu clapeta apăsată a mouse-ului) se inserează, respectiv şterg pictograme din barele de instrumente.

   Exercițiu. Inserați butoanele de subscript şi superscript in bara de instrumente Format.
3. Studiați posibilitățile de utilizare a diacriticelor în document:
   - prin selecție limbaj instalat (Ro dacă există în bara de stare) – identificați tastele corespunzătoare
   - prin utilizarea unui set de caractere care include diacritice (Arial de exemplu) şi pentru care se definesc shortcut-uri)
4. Studiați formatatarea în pagină(File-> Page Setup)
   Exercițiu: - setați un document:
   - format A4
   - margini de 1 cm (dreapta, stănga, sus, jos)
   - antet la 1,2 cm
   - notă de subsol la 1,2 cm.
   - poziționare verticală
5. Salvați pe disc, în folderul corespunzător grupe dvs., documentul cu numele ExercițiuWord1.doc
6. Inspectați seturile de caractere instalate în sistemul dvs. (Control Panel -> Fonts)
   ATENȚIE! Control Panel nu se află în Word.
   Studiați opțiunea Format -> Font pentru schimbarea fonturilor.
   Exercițiu: scrieți un text în care să utilizați patru tipuri de fonturi şi care să includă indici superiori şi inferiori.
7. Studiați formatarea paragrafelor (Format -> Paragraph) şi alinierea acestora (dreapta, stanga, centru, full)
   Exercițiu: - editați patru paragrafe cu formatări şi alinieri diferite.
8. Studiați opțiunea de marcare/numerotare:
   Exercițiu: - includeți în document 4 liste diferit marcate.
9. Studiați modalitățile de inserare antete şi note de subsol (View-> Header and Footer)

Exercițiu: inserați în document antetul facultății dvs. şi ca notă de subsol denumirea documentului şi pagina.

10. Salvați documentul creat
11. Afişați documentul pe 2 coloane.

LABORATOR 3

I. Editorul WORD

1. Aplicatii complementare:
    a. *scriere artistica cu MS WordArt*
    b. editare ştiinţifică MS Equation Editor
2. Utilizarea editorului grafic Draw
3. Editarea documentelor pe coloane (crearea de coloane stil ziar, echilibrarea coloanelor pe pagina) [Format – Columns]
4. Lucrare practică: editare document prezentat de profesor la laborator
5. Studiați modalitățile de realizare a unui tabel utilizând editorul Word (MS Office)

   Opțiunea TABLE:
   - inserare -> Insert Table (cu specificarea nr. linii, nr. coloane)
   - formatare tabel -> Draw Table
   - operații de formatare tabel (linii, coloane, celule):
       i. operații generale: inserare, modificare, copiere, mutare, stergere, aspect chenar (Tables and Borders), aliniere
       ii. operatii speciale:
           1. împărțire celule (Split Cell)
           2. unire celule (Merge Cells)
   - operații de formatare date din tabel sau celule:
       - operații generale: inserare, modificare, copiere, mutare, stergere, aspect, aliniere, schimbare font
   - operații de prelucrare date:
       i. ordonare –> Table Sort (crescătoare, descrescătoare)
       ii. prelucrări numerice -> Table Formula:
           - minim, maxim, medie, modul, rotunjire, numărare, însumare, etc.

6. Lucrare practică: editare document prezentat de profesor la laborator. Ordonați datele după totalul debitor.

III. EXCEL

Studiați modalitățile de realizare a unui tabel utilizând editorul de calcul tabelar Excel (MS Office).

Un document Excel (book), cu extensia **.xls**, conține mai multe foi de lucru (workSheets).

Foaia de lucru este compusă din celule(cells).

STUDIAȚI:

A. Studiați organizarea ecranului la Excel.
B. Organizare Excel:
   - book = document Excel
   - work Sheet = Foaie de lucru
   - celula

- Adresare:
    i. Celula:
        1. relativa: **LiteraNumar** (Litera indica coloana, Cifra indica linia)
        exemplu: A1
        2. absoluta: **$** asociat coloanei si/sau liniei
        exemple: A$1, $A1, A1
    ii. Foaie de calcul: numeFoaie**!**adresaCelula
        Exemple: Sheet1!A1
- formatare celulă (Format Cells)

C. Identificati bara celulei şi elementele acesteia: nume/adresa, editare conținut celulă
D. FORMULE EXCEL
- se introduc cu **=** în față
- cuprind
    i. Expresii formate din operanzi (adrese de celule sau valori) şi operatori numerici (+, -, *, /)
    ii. Funcții Excel: -> Insert/Function

E. Studiați toate facilitățile de la punctul 1, dar în Excel.

LABORATOR 4

I. EXCEL

1. Lucrare practică: editare document prezentat de profesor la laborator.
Se vor realiza următoarele calcule
TOTAL DEBIT= SOLD INITIAL DEBIT +RULAJ DEBIT
TOTAL CREDIT= SOLD INITIAL CREDIT +RULAJ CREDIT

2. Studiu expresie condițională. Funcția IF
Lucrare practică
SOLD FINAL DEBIT= TOTAL DEBIT – TOTAL CREDIT daca val. este pozitivă
SOLD FINAL CREDIT= TOTAL CREDIT – TOTAL DEBIT daca val. este pozitivă

3. Studiu formatare condițională. Opțiune Format -> Conditional Formating
Lucrare practică.
Verificare balanță: Suma DEBIT = Suma CREDIT pe toate perechile DEBIT-CREDIT.
In caz de neegalitate se vor afişa sumere corespunzătoare cu roşu.

4. Ordonați datele din tabel după TOTAL DEBIT. Pentru sume egale pe debit, după TOTAL CREDIT.

5. FORMULE EXCEL
- se introduc cu **=** în față
- cuprind
  Expresii formate din operanzi (adrese de celule sau valori) şi operatori numerici (+, -, *, /)
  Funcții Excel: -> Insert/Function
- se termină cu <Enter>, sau <CTRL><SHIFT><Enter> pentru calcule matriciale

6. Lucrare practică: înmulțirea a două matrici. Matricea rezultat se va poziționa într-o foaie separată de lucru. Se va utiliza funcția MMUL.
   Se va calcula şi determinantul matricei rezultat (funcția MDETERM).

7. Lucrare practică: editarea ştatului de plată. Vezi modelul prezentat.

8. Reprezentări grafice. Să se realizeze graficul pentru variațiile medii USD şi EUR pentru perioada 1985-2002 (vezi site www.bnro.ro).

9. Studiați opțiunile de prelucrare a datelor:
   - sortare
   - filtrare

- subtotal.

10. Studiați funcțiile financiare:
    FV
    PV
    NPER
    PMT
    IPMT

LABORATOR 5

I. Programarea in limbaj de asamblare

INSTRUMENTE utilizate pentru programarea în limbaj de asamblare

1. Adresa Internet pentru limbaje de asamblare:
   http://www.geocities.com/SiliconValley/Lab/1563/assembly.html

2. ETAPE IN REALIZAREA PROGRAMELOR EXECUTABILE:

a) Editarea programului
- se editeaza fisierul sursa al programului care cuprinde succesiunea de instructiuni in limbaj de asamblare care reprezinta algoritmul
- se poate folosi in acest scop orice editor de texte neformatate (Notepad, Edit sau editoarele mediilor de programare)
- fisierul editat trebuie sa aiba extensie *.ASM

b) Asamblarea programului
- se asambleaza (translatare unu la unu a instructiunilor in limbaj masina) programul folosind un asamblor (tasm.exe sau masm.exe). Sintaxa:
  tasm fisier[.asm]
- asamblorul realizeaza o verificare sintactica a fisierului sursa semnalizand erorile de sintaxa
- fisierul rezultat are extensia *.OBJ

c) Linkeditarea(Editarea de legaturi) programului
- se realizeza editarea de legaturi folosind linkeditorul (tlink.exe sau mlink.exe). Sintaxa:
  tlink fisier[.obj]
- fisierul rezultat are extensia *.EXE si reprezinta programul executabil

d) Depanarea programului
- pentru detectia erorilor de logica a programului se foloseste o aplicatie de depanare (td.exe).
- fiserul analizat este cel cu extensie*.EXE
- se realizeaza modificarile in fisierul sursa (pas a) si se reia succesiunea de operatii pentru obtinerea fisierului executabil.

3. INSTRUMENTE UTILIZATE
   a. Editor de texte neformatate: Notepad, editor NC, edit
   b. Asamblor: tasm.exe
   c. Linkeditor: tlink+biblioteci
   d. Depanator: debug, td.exe
4. STUDIUL UTILITARULUI TD (turbo debuger)

Functia principala : mediu prietenos pentru depanarea programelor
Servicii functionale:
- incarcare programe

- afisarea continutului segmentului de cod, dezasamblat
- afisarea continutului segmentului de date
- afisarea continutului registrilor microprocesorului (inclusiv indicatorii de conditie)
- urmarire executie program pas cu pas, animata, cu breakpoints
- inspectare variabile de memorie si evaluare expresii cu acestea
- vizualizare stiva, jurnale de executie, coprocesor matematic, clippboard, etc pe optiunea "View"

Servicii aditionale:
- editare
- acces la fisiere
- configurare context ("Options")
- lucru cu ferestre ("Windows")
- asistenta on-line ("Help")

Se va studia utilitarul urmarind optiunile cu care se apeleaza functiile prezentate mai sus.
Se vor studia si celelalte optiuni.

5. OPERATII DE I/E

Utilizarea functiilor SO DOS. Intreruperea 21h. Se va folosi aplicatia TECHELP.
Studiu:
- se vor studia functiile 01 si 3fH pentru preluarea caracterelor de la tastatură şi, 02 şi 09H, pentru afişarea caracterelor pe ecran (ce valori se incarca in registri pentru apelul functiilor, ce registri se incarca la iesire, unde se depun datele preluate si de unde se iau la afisare)
- se va studia functia 4cH pentru terminarea programului
- se va introduce cu TD secventa de instructiuni necesara utilizarii acestor functii, se va executa si urmari incarcarea memoriei si a registrilor in executie.

6. Studiati capitolele 1 şi 2 din cartea ArtOfAsm.

LABORATOR 6

I. Structura programului sursă

Programul sursa editat in limbaj de asamblare este structurat corespunzator segmentelor de memorie.

Segmentele de memorie sunt definite cu directivele asamblor (.segment pentru inceputul segmentului si .ends pentru sfirsitul segmentului)

<nume> SEGMENT [<tip>]
    <corp segment>
<nume> ENDS

Stuctura programului este:

.model <tip_model_memorie>

.data [segment]
[.data ends]

[.stack <n>]

.code [segment]
end

- **directiva .model** specifica tipul de model de memorie folosit dupa cum urmeaza:

  model **tiny** = in acest model toate segmentele de date, cod, stiva sunt intr-un singur segment de memorie, cu dimensiunea mai mica de 64 koct.; toate adresele de sunt de tip near; adresele sunt relative.

  model **small** = in acest model segmentul de date poate fi separat de segmentul de cod cu dimensiunea mai mica de 64 koct.; toate adresele de sunt de tip near; adresele sunt relative.

  model **medium** = in acest model segmentul de date poate fi mai mare de 64koct, iar segmentul de cod cu dimensiunea mai mica de 64 koct.; toate adresele de sunt de tip near; adresele sunt relative.

  model **compact** = in acest model segmentul de date e mai mic de 64koct, iar segmentul de cod poate depasi 64 koct.;

  model **large** = in acest model segmentul de date poate fi mai mare de 64koct, dar o structura de date nu poate depasi 64koct.; segmentul de cod poate depasi 64 koct.;

  model **huge** = in acest model segmentul de date poate fi mai mare de 64koct; segmentul de cod poate depasi 64 koct.;

II. Declararea datelor

- **directiva .data** precede declaratiile de date si initializarile

forma generala a datelor:
[<nume>] <tip> <lista_expresii>
unde:
nume = prin care este referita data; valoarea este valoarea este adresa la care se gaseste in memorie la referire;
tip = DB – pentru date de tip octet
DW – date de tip cuvant
DD – date de tip pointer (dublu cuvant)
DQ – date de tip virgula mobila, de 8 octeti; folosite pentru reprezentarea numerelor reale
DT – date de 10 octeti pentru reprezentarea numerelor BCD
lista expresii = valorile cu care se initializeaza zona de date rezervate pentru declaratia respectiva;
? indica rezervarea fara initializare.
pentru a initializa o zona de memorie cu aceeasi valoare se poate folosi functia de multiplicare dup cu urmatoarea sintaxa:
<nr.> dup(<valoare>)
unde <nr.> reprezinta factorul de multiplicare, iar <valoare> valoarea care se multiplica.

Initializarea segmentului de date se va face cu instructiunile:
mov ax,@data
mov ds,ax
Instructiunea **end** indica sfarsitul programului.

III. Apeluri de funcții sistem pentru operații de I/E

Studiați în documentația TECHELP, pe întreruperea 21h, următoarele funcții ale sistemului de operare:
- preluare caractere de la tastatură: 01h, 3fh
- afişare caractere pe ecran: 02h, 09h
- terminare normala program: 4ch

2. Exercitiu Se va realiza urmatorul program in limbaj de asamblare, urmarind etapele prezentate mai sus:

```
        .model small
        .stack 100h
        .data
sir     db 80 dup('$')
m1      db 'Primul program in limbaj de asamblare',13,10,'$'
m2      db 'Sirul preluat de la tastatura este: ','$'
        .code
        mov ax,@data
        mov ds,ax
```

```
mov ah,9h
mov dx,offset m1
int 21h

mov bx,0
mov cx,80
mov ah,3fh
mov dx,offset sir
int 21h

mov ah,9h
mov dx,offset m2
int 21h

mov ah,9h
mov dx,offset sir
int 21h

mov ah,4ch
int 21h
end
```

- directiva .code precede instructiunile ce reprezinta codul programului

LABORATOR 7

Moduri de adresare

I. Parte teoretică

Modurile de adresare specifica modalitatea de obtinere a datelor de prelucrat. Sunt prezentate in continuare cele mai utilizate moduri de adresare si exemple de folosire a lor.

1. Adresare imediata – in instructiune apare valoarea operandului

```
.data
var1    db      2
var2    dw      5 dup('ab')
.code

mov al,2                        ; valoarea operandului este in instructiune
mov ax,offset var2   ;valoarea operandului este adresa relativa in segmentul de
date
mov ax,data          ;valoarea operandului este adresa de segment pentru data
```

2. Adresare directa – in instructiune apare adresa operandului

```
mov ax,var2  ; var2 reprezinta adresa de memorie de la care se incarca
operandul
mov ax, word ptr var2+1     ; word ptr specifica dimensiunea word (2 octeti)
                            pentru operand, deci se ;incarca in ax valorile de la adresa
                            var2+1
mov var2+2,ax
mov var2(2),ax
add cx,[100]
```

3. Adresare indirecta prin registri – operandul se gaseste in registru (bx, si sau di)

```
mov ax,[bx]   ;operandul se afla la adresa indicata de registrul bx
mov [di],cx
add byte ptr[si],2 ; instructiune echivalenta cu
        add[si],2       ;(cu obs. ca nu se cunoaste dimensiunea operandului)
```

4. Adresare bazata

Pentru obtinerea adresei operandului se foloseste un registru de baza (bx sau bp), iar adresa stocata in acesta se aduna cu un deplasament. Registrul segment pentru bx este ds, iar pentru bp este ss.

```
.data
var     dw      10 dup(10)
```

```
.code
mov bx,5       ; s-a pregatit registrul de baza

mov ax,var[bx]
mov ax,bx[var]
mov ax,[bx+var]
mov ax,[bx].var
;instructiunile sunt echivalente si transfera in ax al 5-lea element de doi octeti de la adresa var

sau
mov bx,offset var     ; s-a pregatit registrul de baza
mov ax,[bx]
```

5. Adresare indexata

Pentru obtinerea adresei operandului se foloseste un registru index (si sau di), iar adresa stocata in acesta se aduna cu un deplasament. Registrul segment pentru si este ds, iar pentru di este es.
Utilizarea este aceeasi cu adresarea bazata:

```
mov si,5

mov ax,var[si]
```

6. Adresare bazata si indexata – in instructiune apar registri de baza si index utilizati pentru obtinerea adresei operandului. Adresa efectiva se obtine adunind valoarea din registrul de baza cu cea din registrul index si adresa relativa continuta in instructiune.

```
mov ax,var[bx][si]
mov ax,var[bx+si]
mov ax,[bx+di+offset var]
```

II. Parte practică

Sa se realizeze un program, in limbaj de asamblare, care afiseaza caracterul cu codul ASCII cel mai mare dintr-un sir de caractere preluate de la tastatura.

III. Temă

1. Sa se realizeze un program, in limbaj de asamblare, care afiseaza caracterul cu codul ASCII cel mai mic dintr-un sir de caractere preluate de la tastatura.
2. Sa se realizeze un program, in limbaj de asamblare, care afiseaza ordonat crescăror un sir de cifre preluate de la tastatura.

LABORATOR 8

I. Programe de conversie ASCII->zecimal-> binar->zecimal -> ASCII şi operații aritmetice elementare

1. Scrieti un program care sa preia de la tastatura un numar de doua cifre (formatul de preluare este ASCII), il converteste in valoare numerica si il depune in memorie.
2. Scrieti un program care converteste un numar zecimal de doua cifre in format ASCII şi il afiseaza pe ecran
3. Scrieti un program care preia o expresie de forma <numar><operatie><numar> si afiseaza rezultatul pe ecran
   Operatia poate fi + sau *.

LABORATOR 9

I. Programe în limbaj de asamblare. Implementarea structurilor decizionale si repetitive

EXERCITIU:
Scrieti un program P3.asm care sa preia de la tastatura o expresie aritmetica cu 2 operanzi de cate maximum 2 cifre zecimale si una din doua operatii: adunare sau inmultire. Va evalua expresia indicata si va afisa rezultatul sub forma expresie=rezultat.
Inmultirea se va simula prin adunare repetata. Programul urmareste implementarea structurilor decizionala si repetitiva.

SOLUTIE:
Programul P3 se obtine din P2. Comentati programul.
P3.asm

```
.model small
.stack 100h
.data
sir db 80 dup('$')
i db 0
j db 0
k db 0
n dw 0
.code
mov ax,@data
mov ds,ax
mov bx,0
mov dx,offset sir
mov cx,80
mov ah,3fh
int 21h
mov si, offset sir
call nr
mov i,al
inc si
mov al,[si]
mov k,al
inc si
call nr
mov j,al
inc si
mov al,'='
mov [si],al
```

```
;IF K="+" THEN N=I+J
;ELSE IF K="*" THEN N=I*J
;se implementeaza cu urmatoarele secvente
;IF K="+" THEN GOTO Plus
mov al,k
cmp al,'+'
je Plus
;ELSE IF K="*" THEN GOTO Inm
cmp al,'*'
je Inm
;ELSE GOTO sf
jmp sf
Plus:
xor ah,ah
mov al,i
add al,j
daa
jnc Afis
inc ah
jmp Afis
Inm:
;Conversie j din BCD in binar si incarcare in cx
mov al,j
and al,0f0h
shr al,1
mov cl,al
shr al,2
add cl,al
mov al,j
and al,0fh
add cl,al
cont: dec cl
xor ch,ch
xor ah,ah
mov al,i
;Realizare inmultire prin adunari repetate
;FOR nr=1 TO J
; I=I+I
;NEXT nr
repet: add al,i
daa
jnc lrep
inc ah
lrep: loop repet
Afis: mov bx,ax;
add ah,30h
```

```
mov [si+1],ah
and bl,0f0h
shr bl,4
add bl,30h
mov [si+2],bl
and al,0fh
add al,30h
mov [si+3],al
mov dx, offset sir
mov ah,9h
int 21h
sf: mov ah,4ch
int 21h
nr: mov bl,0
mov al,[si]
mov bh,'+'
cmp [si+1],bh
jz gata
mov bh,'*' ;se acopera cazul operatiei de inmultire
cmp [si+1],bh
jz gata
mov bh,0dh ;se acopera cazul ultimului operand cu o
cmp [si+1],bh ;singura cifra, care nu e urmat de semnul '+'
jz gata ;sau '*' ci de codul 0A, corespunzator LF
sub al,30h
shl al,4
mov bl,al
inc si
mov al,[si]
gata: sub al,30h
or al,bl
ret
end
```

LABORATOR 10

I. Declararea şi utilizarea procedurilor în limbaj de asamblare

Procedurile pot fi de două feluri: NEAR sau FAR.
Procedurile NEAR se află în acelaşi segment de cod cu programul în care sunt apelate. In stivă se salvează numai adresa offset a programului scurent.
Procedurile FAR se află în alt segment de cod fată de programul în care sunt apelate. În stiva se salveza adresa de segment de cod şi adresa offset.

Toate procedurile se termină cu instrucțiunea **ret** care predă controlul programului apelant (prin încărcarea registrului IP cu valorile din vărful stivei, ce în mod normal reprezintă adresele salvate la apelul procedurii).

Sintaxa generală pentru definirea procedurilor este:

```
<nume>      proc   [<atribut>]
        <corp procedură>
<nume>      endp
```

unde:
<nume> = numele prin care se face apel la procedură. Numele are asociată ca valoare adresa primei instrucțiuni din procedură.
<atribut> = FAR sau NEAR

Procedurile pot fi imbricate. Adică într-o procedură se poate defini o altă procedură.

Apelul procedurilor se realizează cu instrucțiunea CALL <nume>

II. Utilizarea macroinstrucțiunilor în limbaj de asamblare

Macroinstrucțiunile = nume simbolic asociat unei secvențe de instrucțiuni în limbaj de asamblare.
Apariția unei macroinstrucțiuni este înlocuită automat de către asamblor cu secvența de instrucțiuni asociată.
Diferența față de procedură este că corpul macroinstrucțiunii este pus în program ori de câte ori apare numele ei.

O macroinstrucțiune poate fi definită oriunde în textul programului cu condiția ca definiția să apară înainte de prima utilizare.
Forma generală pentru definirea macroinstrucțiunilor este:

```
<nume> MACRO [<lista de parametrii>]
        <corp macroinstructiune>
    ENDM
```

unde <lista de parametri> este lista de parametri formali în fucție de care este definită macroinstrucțiunea. Parametrii apar în corpul macroinstrucțiunii, fiind substituiți ca valorile parametrilor locali.

De exemplu macroinstrucțiunea aduna:

```
aduna:      MACRO     t1,t2,suma
        mov   ax,t1
        add   ax,t2
        mov   suma,ax
        ENDM
```

Apelul macroinstrucțiunii: Macroinstrucțiunile nu se apelează cu instrucțiunea call. Apelul se face cu numele macroinstrucțiunii.

```
        aduna a,b,c
```

Secvența de instrucțiuni generată este:

```
        mov   ax,a
        add   ax,b
        mov   dx,ax
```

Pentru definirea şi utilizarea macroinstrucțiunilor asamblorul acceptă o formă de pseudoinstrucțiune prin care se asociază o valoare unui simbol. Forma generală a pseudoinstrucțiunii este:

<nume> = <expresie>
unde <nume> este numele căruia i se atribuie valoarea corespunzătoare a expresiei.
In segmentul de date se poate realiza o astfel de atribuire cu declarația EQU

```
<nume>      EQU   <valoare>
```

exemplu

```
numar       MACRO     tip
        valoare = valoare+1
        tip     db      valoare
        ENDM
```

Macroinstrucțiunea realizează o rezervare de tip db cu inițializarea variabilei specificate cu valoare.

1. Să se modifice programul de la laboratorul anterior folosind declarații de proceduri şi pseudoinstrucțiuni.

2. Scrieti un program de ordonare (crescatoare sau descrescatoare) a unui sir de numere introdus de la tastatura. Utilizati un algoritm care sa permita proceduri (macroinstructiuni).

LABORATOR 11

I. Modul grafic

Ecranul poate fi programat în două moduri distincte de lucru:

- **text** în care se afişează pe ecran caractere ASCII, fiecare într-o matrice de pixeli
- **grafic** în care se pot afişa grafice şi texte

Standardele pentru afişare sunt:

- CGA (**C**olor **G**raphics **A**dapter) - Interfata pentru monitor grafic color cu urmatoarele caracteristici:

- afiseaza textele pe 25 de rinduri a 80 coloane/40 coloane;
- fiecare caracter este desenat intr-o matrice de 64 de puncte (8x8);
- rezolutie in mod grafic - medie-320x200 puncte x 4 culori
  - mare -640x200 puncte x 2 culori;
- frecventa de lucru a adaptorului este de 14 MHz (la rezolutie mare) si 7MHz (la rezolutie mica).

- EGA (**E**xtended **G**raphics **A**dapter) - Interfata pentru monitor grafic color EGA cu urmatoarele caracteristici:

- afisare texte pe 25 de rinduri a 80/40 de coloane sau 43 rinduri;
- desenul caracterelor e format intr-o matrice de 112 puncte (7x16);
- rezolutie grafica - 320x200 puncte in 16 culori
  - 640x200 puncte in 16 culori
  - 640x350 puncte monocrom
  - 640x350 puncte in 16 culori din 64 posibile;

- VGA (**V**ideo **G**raphics **A**dapter) - Interfata pentru monitor grafic color VGA - Compatibil cu precedentele, dar cu rezolutie mai mare.

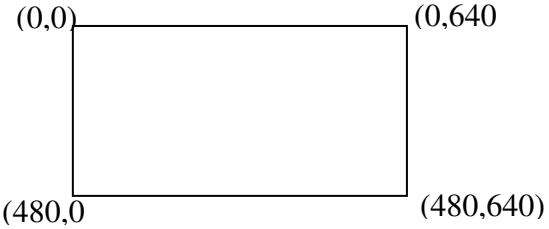
- rezolutia grafica este pina in **640x480** puncte in **16**/**256** culori din 256 culori.
- afisarea in mod text cu o rezolutie de 720x400 puncte, caracterele fiind afisate intr-o matrice de 9x16 puncte;
- cele mai noi cu o rezolutie grafica de **800x600** puncte in **16** culori din 256, text de 132 coloane;

- SVGA (**S**uper **V**ideo **G**raphics **A**dapter) - Interfata pentru monitor grafic color VGA -Compatibil cu precedentele, cu rezolutie mai mare:

- rezolutie grafica pina in **1024x768** puncte in **256** culori
- frecventa de lucru de 40 MHz.

Caracterele sau graficele sint afisate pe linii, de la stinga la dreapta ecranului si de sus in jos. Pozitia curenta de afisare este indicata de un **cursor**, care se muta automat, odata cu afisarea caracterelor.

Coordonatele punctului din st>nga sus sunt (0,0), iar din dreapta jos de rezoluție maximă. De exemplu pentru modul de lucru VGA 640x480:



Întreruperea 10h cuprinde servicii video.

Practic:

1. Studiu INT 10h. Se va identifica funcția de schimbare a modului de lucru de afişare. Se va identifica funcția cu care se aprinde un pixel pe ecran.
2. Studiati urmatorul program

```
.model small
.stack 100h
.data
        a dw 0
        b dw 0
        c dw 0
        d dw 0
.code
        mov ax,@data
        mov ds,ax

        mov ah,0h
        mov al,13h
        int 10h

        mov ah ,01h
        int 21h

        mov a,50
        mov b,150
        mov d,100
        mov c,210

ciclu_1:
        mov ah,0ch
        mov bh,0h
        mov dx,a
        mov cx,b
        mov al,5
        int 10h

        mov ah,0ch
        mov bh,0h
        mov dx,a
        mov cx,c
        mov al,2
        int 10h

        inc a
        cmp a,150
```

```
        jne ciclu_1


        mov a,50
        mov b,150
        mov d,150
        mov c,210

ciclu_2:
        mov ah,0ch
        mov bh,0h
        mov dx,a
        mov cx,b
        mov al,3
        int 10h

        mov ah,0ch
        mov bh,0h
        mov dx,d
        mov cx,b
        mov al,4
        int 10h

        inc b
        cmp b,210
        jne ciclu_2


        mov ah,01h
        int 21h

        mov ah,0h
        mov al,3h
        int 10h

        mov ah,4ch
        int 21h
end
```

3. Tema. Realizati un program care afiseaza un desen pe ecran.

LABORATOR 12

PREGATIRE PROIECT

LABORATOR 13

TEST LABORATOR

LABORATOR 14

SUSTINERE PROIECT

ÎNTREBĂRI TESTE

TESTE LABORATOR

1. Descrieti apelurile de functii sistem utilizate in programele din laboratoare.
2. Care sunt etapele parcurse in programarea in limbaj de asamblare?
3. Cum se numeste fisierul care contine programul scris in limbaj de asamblare?
4. Ce extensie are fisierul care contine programul scris in limbaj de asamblare?
5. Care este sintaxa comenzii de asamblare a unui program p1?
6. Ce erori sunt semnalate la asamblare?
7. Ce extensie are fisierul generat de asamblor?
8. Fisierul executabil se obtine in urma operatiei de.....
9. Pentru ce se utilizeaza td.exe?
10. Ce erori se detecteaza cu debuger-ul?
11. Care este structura programului scris in limbaj de asamblare?
12. Care este sintaxa generala de declaratie a unui segment de memorie?
13. Ce caracterizeaza modelul tiny de memorie?
14. Ce caracterizeaza modelul small de memorie?
15. Ce caracterizeaza modelul compact de memorie?
16. Ce caracterizeaza modelul large de memorie?
17. Ce caracterizeaza modelul huge de memorie?
18. Ce caracterizeaza modelul medium de memorie?
19. Care este sintaxa generala a declaratiilor de date in memorie?
20. Ce tipuri de date se pot declara in segmentul de date?
21. Ce reprezinta ? in declaratiile de date
22. Care este continutul memoriei pentru urmatoarea declaratie de date:
    var DB 1, ‘1’, 2 dup(‘a’)
23. Care sunt instructiunile utilizate pentru initializarea segmentului de date?
24. Fie sirul:

Mes DB ‘test$

Care este succesiunea de instructiuni cu care se face afisarea lui pe ecran?

25. Ce greseli sunt in secventa urmatoare de program:

```
.data
mes     DB   ‘test’
...
.code
mov ah,9
mov dx,mes
int 21h
...
```

26. Care este succesiunea de instructiuni pentru preluarea unui sir de caractere de la tastatura?
27. Ce functie sistem se apeleza la sfarsitul programului?

28. Ce este adresarea imediata? Exemple
29. Ce este adresarea directa? Exemple
30. Ce este adresarea indirecta? Exemple
31. Ce este adresarea bazata? Exemple
32. Ce este adresarea indexata? Exemple
33. Ce este adresarea bazata si indexata? Exemple
34. Se considera urmatoarea secventa de program:

```
        sir     db      '12574893'
        .....

        xor ah,ah
        mov cl,8
        mov si,offset sir
iar:    mov al,[si]
        cmp al,ah
        jl et
        mov ah,al
et:     inc si
        dec cl
        jnz iar
```

Ce valoare se incarca in registrul ah? 2 pct.

35. Se considera urmatoarea secventa de program:

```
        sir     db      2,5,9,3,4,1,5
        .....


        mov si,offset sir
                mov cl,7
        mov ah,[si]
urm:    mov al,[si]
        cmp al,ah
        jg et
        mov ah,al
et:     inc si
        dec cl
        jnz urm
```

36. Scrieti succesiunea de instructiuni care implementeaza adunarea a doua numere i si j stocate in memorie in format BCD si depunerea rezultatului in registrul ax
37. Scrieti succesiunea de instructiuni care realizeaza conversia unui numar j stocat in memorie in format BCD impachetat in valoare binara stocata in registrul cx.
38. Sa se scrie succesiunea de instructiuni ce implementeaza operatia de inmultire a unei valori i stocate in memorie in format BCD intr-un octet si a unei valori binare stocate in registrul cx. Rezultatul se va depune in registrul ax.

39. Se se scrie succesiunea de instructiuni pentru conversia unui numar BCD impachetat din registrul al in cod ASCII necesar afisarii.
40. Se se scrie succesiunea de instructiuni pentru conversia a doua caractere ASCII stocate in memorie la adresa sir in cod BCD impachetat stocat in registrul al.
41. Care este valoarea binara a lui 268?
42. Care este valoarea zecimala a numarului hexazecimal e6?
43. Sa se exprime 111100111 hexazecimal şi zecimal.
44. Sa se scrie secventa de instructiuni pentru preluarea unui sir de caractere de la tastatura.
45. Sa se scrie secventa de instructiuni pentru afisarea unui sir de caractere pe ecran.
46. Sa se scrie secventa de instructiuni pentru preluarea unui sir de caractere de la tastatura.
47. Care sunt etapele realizarii programelor executabile?
48. Care este sintaxa comenzii de asamblare si ce operatie realizeaza acesta?
49. Care este sintaxa comenzii de linkeditare si ce operatie realizeaza acesta?
50. Care este structura generala a unui program scris in limbaj de asamblare?
51. Ce operatii implementeaza urmatoarea secventa de program:

```
mov bh,'+'
cmp [si+1],bh
jz gata
mov bh,'*'
cmp [si+1],bh
jz gata
mov bh,0dh
cmp [si+1],bh
jz gata
```

52. Ce operatie implementeaza urmatoarea secventa de program:

```
mov al,[si]
sub al,30h
shl al,4
mov bl,al
inc si
mov al,[si]
sub al,30h
or al,bl
```

53. Ce operatie implementeaza urmatoarea secventa de program

```
mov al,j
and al,0f0h
shr al,1
mov cl,al
```

```
        shr al,2
        add cl,al
        mov al,j
        and al,0fh
        add cl,al
```

54. Ce operatie implementeaza urmatoarea secventa de program:

```
        xor ah,ah
        mov al,i
repet:  add al,i
        daa
        jnc lrep
        inc ah
lrep:   loop repet
```

55. Ce operatie implementeaza urmatoarea secventa de program:

```
        mov bx,ax;
        add ah,30h
        mov [si+1],ah
        and bl,0f0h
        shr bl,4
        add bl,30h
        mov [si+2],bl
        and al,0fh
        add al,30h
        mov [si+3],al
```

56. Care este sintaxa generala de definire a procedurilor?
57. Care este sintaxa generala de definire a unei macroinstructiuni?

58. Gasiti greseala din urmatoarea secventa de program si corectati-o.

```
aduna:      macro t1,t2,suma
        mov ax,t1
        add ax,t2
        mov suma,ax
endm
....
call aduna
```

59. Descrieti formatul “com” pentru fisiere executabile.
60. Descrieti formatul “exe” pentru fisiere executabile.
61. Ce sunt programele TSR?
62. Ce standarde pentru afisare cunoasteti?
63. Ce operatie implementeaza urmatoarea secventa de program:

```
        mov ah,0
        mov al,13h
```

```
        int 10h
```

64. Ce operatie implementeaza urmatoarea secventa de program:

```
        mov bl,50
        mov bh,0
        mov cx,100
        mov dx,100
iar:    mov ah,0ch
        mov al,5
        int 10h
        inc cx
        inc dx
        dec bl
        jnz iar
```

65. Ce operatie implementeaza urmatoarea secventa de program:

```
next:   mov al,[si]
        cmp bl,al
        jg ok
        mov bl,al
ok:     inc si
        dec cl
        jnz next
```

66. Sa se scrie un program in limbaj de asamblare care extrage valoarea maxima dintr-un sir de numere intregi stocate in memorie
67. Sa se scrie un program in limbaj de asamblare care extrage valoarea minima dintr-un sir de numere intregi stocate in memorie
68. Sa se scrie un program in limbaj de asamblare care afiseaza suma a doua numere stocate in memorie.
69. Sa se scrie un program in limbaj de asamblare care cauta un caracter dat intr-un sir de caractere preluate de la tastatura.
70. Sa se scrie un program in limbaj de asamblare care afiseza o dreapta oblica pe ecran.
71. Sa se scrie un program in limbaj de asamblare care afiseaza un patrat pe ecran.

ÎNTREBĂRI GRILĂ CURS

1 Ce este “cutia neagra"? Care este cea mai importanta proprietate a ei?
2 Calculatorul numeric rezolva problemele prin:
  a) folosirea unui limbaj de programare
  b) folosirea unor cutii negre
  c) executia unor instructiuni
3 In arhitectura multinivel a calculatorului, fiecare nivel este considerat:
  a) o abstractizare care ofera detalii de implementare nivelului superior
  b) o abstractizare care nu ofera detalii de implementare nivelului superior
  c) un nivel de circuite integrate
4 Ce este masina virtuala? Cum se numeste limbajul de programare a acesteia?
5 Prin ce metoda executa programele unui nivel superior?
  a) prin translatare
  b) prin interpretare
  c) prin executia instructiunilor fara nici o transformare
6 Ce este translatarea?
7 Ce este interpretarea?
8 Puneti in ordinea fireasca urmatoarele niveluri abstracte (numerotati-le de sus in jos)
  Nivelul limbajului de asamblare
  Nivelul logic digital
  Nivelul limbajului orientat pe problema
  Nivelul arhitecturii setului de instructiuni
  Nivelul microarhitecturii
  Nivelul masina al sistemului de operare
9 Ce obiecte se gasesc la nivelul logic digital?
10 Ce obiecte se gasesc la nivelul microarhitecturii?
11 Instructiunile nivelului ISA (Instruction Set Architecture) sunt executate:
  a) prin interpretare
  b) direct de catre hardware
  c) prin software
12 Asamblorul este un:
  a) interpretor
  b) translator
  c) un program care executa programele scrise in limbaj de asamblare
  d) un program cu care se scriu programele in limbaj de asamblare
13 Ce reprezinta partea hardware a calculatorului?
14 Ce reprezinta partea software a calculatorului?
15 Exista o granita clara intre hardware si software referitor la inplementarea unor anumite functi
  Justificati raspunsul.
  a) da
  b) nu
16 Care sunt elementele principale componente ale arhitecturii von Neumann?

17 Ce rol are unitatea centrala a calculatorului?
18 Din ce este alcatuita unitatea centrala a calculatorului?
19 Executia programului (alcatuit din instructiuni) si controlul activitatii intregului sistem este realiz
   a) unitatea aritmetica si logica
   b) unitatea de comanda
   c) unitatea centrala
20 Registrii sunt folositi pentru stocarea temporara a urmatoarelor tipuri de date de prelucrat:
   a) date
   b) adrese
   c) starea programului
   d) gestiunea memoriei
21 Memoria stocheza datele in format:
   a) zecimal
   b) hexazecimal
   c) binar
   d) octal
22 Memoria este alcatuita din:
   a) adrese
   b) locatii de memorie
   c) biti
23 Care sunt operatiile pe care le poate executa memoria?
24 Memoria RAM este :
   a) memoria primara a sistemului
   b) memoria de masa
   c) memoria cache
25 Care sunt criteriile de aranjare a tipurilor de memorii in ierarhia acestora?
26 Memoria indispensabila sistemului de calcul este:
   a) memoria de masa
   b) memoria cache
   c) memoria primara
27 Cea mai rapida forma de memorie o reprezinta:
   a) registrii microprocesorului
   b) memoria cache
   c) memoria primara
28 Ce sunt memoriile cache?
29 Ce sunt bufferele?
30 Ce reprezinta magistralele?
31 Magistralele se clasifica in functie de tipul de informatie care circula prin ele in?
32 Ce informatie circula prin magistrala interna a microprocesorului?
33 Enuntati Legea lui Moore.
34 Ce exprima legea lui Moore?
   a) cresterea preturilor
   b) evolutia programelor
   c) progresul tehnologic

35 Enuntati legea lui Nathan Myhrvold
36 Ce exprima legea lui Nathan Myhrvold?
   a) evolutia tehnologica
   b) cerinta software-ului permanenta de resurse hardware
   c) implementarea software-ului
37 Primul microprocesor realizat de firma Intel a fost:
   a) Intel8008
   b) Intel4004
   c) Intel8080
38 Incercuiti caracteristicile microprocesoarelor pe 8 biti(1973-1977):
   a) set de instructiuni pentru manipularea datelor pe 8 biti
   b) set de instructiuni pentru manipularea datelor pe 16 biti
   c) stiva implementata hardware
   d) stiva in memorie implementata software
   e) limbajul de programare=limbajul de nivel inalt
   d) limbajul de programare=limbajul de asamblare
39 Care sunt unitatile functionale ale microprocesoarelor actuale:
   a) CPU
   b) FPU
   c) MMU
   d) MMX
40 Ce functii realizeaza CPU?
41 Ce functii realizeaza FPU?
42 Ce functii realizeaza MMU?
43 Ce functii realizeaza MMX?
44 Care sunt cele doua directii arhitecturale de dezvoltare a microprocesoarelor?
45 Ce sunt calculatoarele personale?
46 Ce sunt calculatoarele de tip server?
47 Ce sunt supercalculatoarele?
48 Ce reprezinta NOW si COW
49 Ce este microprocesorul?
50 Care, din urmatoarele, reprezinta componente functionale ale microprocesorului:
   a) unitatea aritmetica si logica
   b) unitatea de comanda
   c) registrii
   d) memoria program
   e) memoria de date
51 Registrul tampon de adrese este cel prin care:
   a) se transmit adresele catre memorie
   b) se primesc adresele de la memorie
   c) se depun adresele de memorie pentru o folosire ulterioara
52 Registrul tampon de date este cel prin care:
   a) se transmit datele catre memorie
   b) se primesc datele de la memorie

c) se depun datele pentru o folosire ulterioara

53 Semnalele de comanda sunt semnalele electrice prin care microprocesorul:
a) culege informatii din sistem
b) transmite informatii de stare in sistem
c) da comenzi de executie componentelor
d) primeste comenzi de executie de la componente

54 Semnalele de stare sunt semnalele electrice prin care microprocesorul:
a) culege informatii din sistem
b) transmite informatii de stare in sistem
c) da comenzi de executie componentelor
d) primeste comenzi de executie de la componente

55 Bifati activitatile asociate unitatii de comanda:
a) programeaza executia secventiala a operatiilor necesare efectuarii instructiunilor
b) dirijeaza fluxul de date
c) coreleaza viteza de lucru a unitatii centrale cu memoria
d) realizeza operatii aritmetice sau logice cu numere intregi

56 Masura numarului de biti ai microprocesorului este data de:
a) latimea magistralei de adrese
b) latimea magistralei de date
c) lungimea registrilor interni

57 Aranjati in ordinea de executie operatiile efectuate de microprocesor pentru exec.unei instruct
a) instructiunea este copiata in registrul de instructiuni
b) adresa instructiunii de executat se copiaza din reg. contor al programului in reg. de adrese
c) continutul memoriei adresate se incarca in registrul de date
d) instructiunea este decodificata
e) codul operatiei se aplica decodificatorului de instructiuni
f) registrul contor de instructiuni se actualizeaza cu adresa urmatoarei instructiuni de executat
g) unitatea de control genereaza semnalele necesare executiei instructiunii

58 Unitatea aritmetica si logica este un circuit combinational cu:
a) doua iesiri si o intrare
b) doua intrari si doua iesiri
c) doua itrari si o iesire

59 Contorul de instructiuni (PC) este un registru care stocheza:
a) adresa instructiunii in executie
b) adresa urmatoarei instructiuni de executat
c) adresa urmatoarei date folosite in program

60 Continutul registrului contor al programului (PC):
a) seautodecrementeaza
b) se autoincrementeza
c) este cu prescriere

61 Instructiunile sunt vehiculate prin:
a) registrul de adresare a memoriei
b) registrul de date (de I/O)
c) registrul de stare

62 Registrul de instructiuni pastreaza instructiunea:
   a) pe durata executarii instructiunii
   b) inainte de a fi executata
   c) dupa ce s-a executat
63 Indicatorii de conditie din registrul de stare se pozitioneaza dupa:
   a) operatii de transfer de date
   b) operatii aritmetice
   c) operatii logice
64 Indicatorii de conditie sunt utilizati in implementarea structurilor program:
   a) alternative
   b) repetitive
   c) liniare
65 Flagul CARRY indica:
   a) existenta unui transport de la cel mai semnificativ bit al rezultatului
   b) existenta unui imprumut la cel mai semnificativ bit al rezultatului
   c) existenta unui transport de la cel mai putin semnificativ bit al rezultatului
   d) existenta unui imprumut la cel mai putin semnificativ bit al rezultatului
66 Flagul ZERO indica:
   a) valoarea zero pentru cel mai semnificativ bit al rezultatului
   b) valoarea zero pentru cel mai putin semnificativ bit al rezultatului
   c) valoarea zero pentru toti bitii rezultatului
67 Flagul SIGN indica:
   a) valoarea pentru cel mai semnificativ bit al rezultatului (de semn)
   b) valoarea pentru cel mai putin semnificativ bit al rezultatului (de semn)
   c) valoarea corespunzatoare marimii rezultatului
68 Ciclul masina esential in executia instructiunilor este:
   a) ciclul de citire din memorie
   b) ciclul de scriere in memorie
   c) ciclul de extragere-decodificare-executie
   d) ciclul de asteptare
69 Adresarea memoriei se face folosind:
   a) registrii cu scop general
   b) registrul de stare
   c) registrii segment
   d) registrii offset
70 Cum se obtine la I8086 adresa efectiva de memorie pe 20 biti?
71 Care sunt registrii segment?
72 Care sunt registrii offset?
73 Care sunt registrii cu scop general si ce dimensiuni au acestia (la Intel)?
74 Indicatorii de conditie de control sunt:
   a) Carry
   b) Overflow
   c) Interrupt
   d) Sign

e) Direction
f) Parity
g) trap
h) Auxiliarry carry

75 Pentru ajustarile zecimale se foloseste flagul:
a) carry
b) overflow
c) auxiliarry carry

76 Flagul Interrupt indica:
a) aparitia unei intreruperi
b) activarea sistemului de intreruperi
c) dezactivarea sistemului de intreruperi

77 Flagul Direction se refera la:
a) operatiile aritmetice sau logice
b) operatiile cu siruri de caractere
c) operatiile pentru controlul procesorului
d) opearatiile de transfer

78 Mnemonica generala a instructiunilor este:
a) nume_instructiune sursa,destinatie
b) nume_instructiune destinatie,sursa
c) nume_instructiune valoare,destinatie

79 Care sunt clasele de instructiuni care fac parte din grupa instructiunilor pentru transfer date/

80 Care observatii sunt valabile pentru instructiunile de tip mov?
a) afecteaza indicatorii de conditie
b) nu afecteaza indicatorii de conditie
c) permit transferuri in registrii segment
d) operanzii trebuie sa fie de acelasi tip

81 Formatul instructiunilor cuprinde:
a) patru campuri
b) trei campuri
c) doua campuri
d) un camp

82 Instructiunea mov word ptr [bx],0:
a) incarca registrul bx cu valoarea 0
b) incarca in locatia adresata de bx valoarea 0 pe un octet
c) incarca in locatia adresata de bx valoarea 0 pe un cuvant

83 La executia instructiunii **push reg** se executa urmatoarele operatii:
a) se decrementeaza SP cu 2 si se transfera valoarea din reg in varful stivei indicate de SP
b) se incrementeaza SP cu 2 si se transfera valoarea din reg in varful stivei indicate de SP
c) se transfera valoarea din reg in varful stivei indicate de SP si se decrementeaza SP cu 2
d) se transfera valoarea din reg in varful stivei indicate de SP si se incrementeaza SP cu 2

84 La executia instructiunii **pop reg** se executa urmatoarele operatii:
a) se decrementeaza SP cu 2 si se transfera valoarea din reg in varful stivei indicate de SP
b) se incrementeaza SP cu 2 si se transfera valoarea din reg in varful stivei indicate de SP

c) se transfera valoarea din reg in varful stivei indicate de SP si se decrementeaza SP cu 2
d) se transfera valoarea din reg in varful stivei indicate de SP si se incrementeaza SP cu 2

85 Implementati instructiunea **xchg bx,cx** folosind stiva

86 Implementati instructiunea **xchg bx,cx** folosind instructiuni de tip **mov**

87 Instructiunea **in al,71h:**
a) incarca in registrul **al** valoarea 71h
b) citeste in registrul **al** un octet de la portul 71h
c) scrie valoarea din registrul **al** la portul 71h

88 Instructiunea out **71h,al**
a) incarca in registrul **al** valoarea 71h
b) citeste in registrul **al** un octet de la portul 71h
c) scrie valoarea din registrul **al** la portul 71h

89 Registrul dx se foloseste in operatiile de I/E daca:
a) adresa portului este mai mica decat 255
b) adresa portului este mai mare decat 255
c) se transfera date mai mari de un octet

90 Cu ce instructiuni se incarca registrii segment?

91 Instructiunea **lahf** este utilizata pentru:
a) a salva indicatorii de conditie in registrul **ah**
b) a seta indicatorii de conditie
c) a incarca registrul ah cu zero

92 Instructiunea **sahf** este utilizata pentru:
a) a salva indicatorii de conditie in registrul **ah**
b) a seta indicatorii de conditie
c) a incarca registrul ah cu zero

93 Instructiunea **popf** este utilizata pentru:
a) a salva indicatorii de conditie in stiva
b) a seta indicatorii de conditie
c) a scoate valoarea din varful stivei intr-un registru specificat

94 Instructiunea **pushf** este utilizata pentru:
a) a salva indicatorii de conditie in stiva
b) a seta indicatorii de conditie
c) a scoate valoarea din varful stivei intr-un registru specificat

95 Numerele intregi sunt reprezentate in memorie prin:
a) numere binare fara semn
b) numere binare cu semn in complement fata de unu
c) numere binare cu semn in complement fata de doi

96 Codificarea BCD este utilizata pentru a reprezenta:
a) numere binare
b) numere zecimale
c) numere hexazecimale

97 In formatul BCD impachetat se pot reprezenta:
a) numere negative
b) numere pozitive

98 Instructiunea adc dest,sursa realizeaza operatia:
a) sursa<--dest+sursa
b) dest<--dest+sursa
c) dest<--dest+sursa+carry
d) sursa<--dest+sursa+carry

99 Instructiunea add dest,sursa realizeaza operatia:
a) sursa<--dest+sursa
b) dest<--dest+sursa
c) dest<--dest+sursa+carry
d) sursa<--dest+sursa+carry

100 Instructiunea **aaa** realizeaza:
a) o corectie la ASCII la inmultire
b) o corectie BCD la adunare
c) o corectie ASCII la adunare
d) o corectie BCD la inmultire

101 Instructiunea **daa** realizeaza:
a) o corectie la ASCII la inmultire
b) o corectie BCD la adunare
c) o corectie ASCII la adunare
d) o corectie BCD la inmultire

102 Instructiunea **aam** realizeaza:
a) o corectie la ASCII la inmultire
b) o corectie BCD la adunare
c) o corectie ASCII la adunare
d) o corectie BCD la inmultire

103 Instructiunea **aas** realizeaza:
a) o corectie la ASCII la scadere
b) o corectie BCD la adunare
c) o corectie ASCII la adunare
d) o corectie BCD la scadere

104 Instructiunea **das** realizeaza:
a) o corectie la ASCII la scadere
b) o corectie BCD la adunare
c) o corectie ASCII la adunare
d) o corectie BCD la scadere

106 Instructiunea sub dest,sursa realizeaza operatia:
a) sursa<--dest+sursa
b) dest<--dest-sursa
c) dest<--dest-sursa
d) sursa<--dest-sursa

107 Instructiunea subb dest,sursa realizeaza operatia:
a) sursa<--dest-sursa
b) dest<--dest-sursa
c) dest<--dest-sursa+carry

d) dest<--dest-sursa-carry

108 Instructiunea sbb dest,sursa realizeaza operatia:
a) sursa<--dest-sursa
b) dest<--dest-sursa
c) dest<--dest-sursa+carry
d) dest<--dest-sursa-carry

109 Instructiunea cmp dest,sursa realizeaza operatia:
a) compara operanzii sursa si destinatie prin aplicarea functei logice "si"
b) compara operanzii sursa si destinatie prin aplicarea functei logice "sau"
c) compara operanzii sursa si destinatie prin scadere

110 Care este sintaxa corecta a instructiunii dec?
a) dec dest,sursa
b) dec dest
c) dec

111 La instructiunile de inmultire rezultatul se obtine in:
a) registrul AL
b) registrul AX
c) numai in registrul DX
d) registrul AX sau AX si DX

112 La instructiunile de inmultire un operand se afla in:
a) registrul AL
b) registrul AX
c) registrul DX

113 La instructiunile de impartire catul se obtine in:
a) registrul AL
b) registrul AX
c) registrul DX

114 La instructiunile de impartire restul se obtine in:
a) registrul AL
b) registrul AX
c) registrul DX
d) registrul AH

115 Care este sintaxa corecta a instructiunii not?
a) not dest,sursa
b) not dest
c) not

116 Daca in registrul AL este stocata valoarea 11001001, dupa instructiunea **shl al,3** se va obtine
a) 00011001
b) 01001000
c) 001001000
d)11001000

117 Daca in registrul AL este stocata valoarea 11001001, dupa instructiunea **shr al,3** se va obtine
a) 00011001
b) 01001000

c) 000100100
d)11001000

118 Explicati care este diferenta dintre shiftarile aritmetice si cele logice

119 Daca in registrul AL este stocata valoarea 11001001, dupa instructiunea **rol al,3** se va obtine:
a) 01001010
b) 01001110
c) 00111001
d)11001000

120 Daca in registrul AL este stocata valoarea 11001001, dupa instructiunea **ror al,3** se va obtine
a) 01001010
b) 01001110
c) 00111001
d)11001000

121 Care este starea indicatorului carry dupa instructiunea **rcl al,2** daca in AL se afla 00100101?
a) 0
b) 1

122 Care este starea indicatorului carry dupa instructiunea **rcr al,2** daca in AL se afla 00100101?
a) 0
b) 1

123 Daca in AL avem 11011110, in urma instructiunii **and al,0fh** se obtine valoarea:
a) 11011111
b) 00001110
c) 11010001

124 Daca in AL avem 11011110, in urma instructiunii **or al,0fh** se obtine valoarea:
a) 11011111
b) 00001110
c) 11010000

125 Daca in AL avem 11011110, in urma instructiunii **xor al,0fh** se obtine valoarea:
a) 11011111
b) 00001110
c) 11010001

126 Daca in AL avem 11011110, in urma instructiunii **test al,0fh** se obtine valoarea:
a) 11011111
b) 00001110
c) 11010001
d) 11011110

127 La instructiunile cu pentru manipularea sirurilor:
a) sirul sursa este pointat de DS:SI
b) sirul destinatie e pointat de DS:SI
c) sirul sursa e pointat de ES:DI
d) sirul destinatie e pointat de ES:DI

128 Se poate seta sensul de parcurgerea sirurilor de caractere?
a) da
b) nu

c) uneori

129 Contorul pentru instructiunile cu siruri se afla in registrul:
a) AX
b) BX
c) CX
d) DX

130 Cum se actualizeaza registrii la executia instructiunilor cu siruri de caractere?
a) SI, DI se incrementeaza, iar CX se decrementeaza
b) SI, DI se decrementeaza, iar CX se decrementeaza
c) SI, DI se incrementeaza, iar CX se incrementeaza
d) SI, DI se decrementeaza, iar CX se incrementeaza

131 Ce operatie realizeza instructiunea MOVB?

132 Ce operatie realizeza instructiunea MOVW?

133 Ce operatie realizeza instructiunea CMPB?

134 Ce operatie realizeza instructiunea CMPW?

135 Ce operatie realizeza instructiunea SCAB?

136 Ce operatie realizeza instructiunea SCAW?

137 Ce operatie realizeza instructiunea LODB?

138 Ce operatie realizeza instructiunea LODW?

139 Ce operatie realizeza instructiunea STOB?

140 Ce operatie realizeza instructiunea STOW?

141 Aveti urmatoarea secventa de program:

```
        mov ax,100h
        call s1
et1:  ….
s1:  inc ax
        push ax
        ret
```

Dupa executia subrutinei s1 se continua executia de la:
a) et1
b) 100h
c) 101h

142 Instructiunea **CALL NEAR et** realizeaza:
a) un transfer la et in alt segment de cod
b) un transfer la et in segmentul curent de cod
c) un salt la et in segmentul curent de cod
d) un salt la et in alt segment de cod

143 Instructiunea **CALL FAR et** realizeaza:
a) un transfer la et in alt segment de cod
b) un transfer la et in segmentul curent de cod
c) un salt la et in segmentul curent de cod
d) un salt la et in alt segment de cod

144 Instructiunea **jmp et** realizeaza:
a) un salt neconditionat la et

b) un salt conditionat la et
c) apelul rutinei et

145 Ce operatii se realizeaza la executia instructiunii CALL?

146 Ce operatii se realizeaza la executia instructiunii RET?

147 Instructiunea **JZ et** realizeaza saltul la et daca:
a) flagul Zero este 1
b) flagul Zero este 0
c) flagul Sign este 1
d) flagul Sign este 0

148 Instructiunea **JNZ et** realizeaza saltul la et daca:
a) flagul Zero este 1
b) flagul Zero este 0
c) flagul Sign este 1
d) flagul Sign este 0

149 Instructiunea **JC et** realizeaza saltul la et daca:
a) flagul Zero este 1
b) flagul Zero este 0
c) flagul Carry este 1
d) flagul Carry este 0

150 Instructiunea **JNC et** realizeaza saltul la et daca:
a) flagul Zero este 1
b) flagul Zero este 0
c) flagul Carry este 1
d) flagul Carry este 0

151 Instructiunea **JCXZ et** realizeaza saltul la et daca:
a) flagul Zero este 1
b) flagul Zero este 0
c) flagul Carry este 0
d) registrul CX este 0

152 Instructiunile LOOP implementeaza bucle cu testul:
a) la inceput
b) la sfarsit
c) numar finit de pasi

153 Instructiunea corespunzatoare etichetei et din **LOOP et** se afla:
a) inaintea instructiunii LOOP
b) dupa instructiunea LOOP
c) nu se afla in programul respectiv

154 Instructiunea **LOOPZ e**t face saltul la et daca:
a) CX=0 si flagul Zero=0
b) CX<>0 si flagul Zero=0
c) CX=0 si flagul Zero=1
b) CX<>0 si flagul Zero=1

155 Instructiunea INT nr. are ca efect:
a) lansarea rutinei de intrerupere nr. la aparitia unui eveniment extern

b) lansarea rutinei de intrerupere nr.
c) intreruperea programului curent

156 Ce operatii se realizeaza la executia instructiunii INT?
a) salvare in stiva a tuturor registrilor
b) salvare in stiva a adresei de intrerupere a programului
c) salvare in stiva a registrului de stare

157 Ce operatii se realizeaza la executia instructiunii RETI?
a) refacere registri
b) scoatere in stiva a adresei de intrerupere a programului
c) refacere registru de stare

158 Daca in tabela vectorilor de intrerupere se afla la adresa 0000, la ce adresa se afla rutina ape
cu instructiunea INT 3?
a) 0000
b) 0006
c) 000Ch
d) 000Fh

159 Instructiunea CLI:
a) activeaza sistemul de intreruperi
b) dezactiveaza sistemul de intreruperi
c) sterge intreruperea curenta

160 Instructiunea STI:
a) activeaza sistemul de intreruperi
b) dezactiveaza sistemul de intreruperi
c) sterge intreruperea curenta

161 Instructiunea STC:
a) activeaza sistemul de intreruperi
b) dezactiveaza sistemul de intreruperi
c) pune carry pe 0
d) pune carry pe 1

162 Instructiunea CLC:
a) activeaza sistemul de intreruperi
b) dezactiveaza sistemul de intreruperi
c) pune carry pe 0
d) pune carry pe 1

163 Ce efect are instructiunea HLT?

164 Ce efect are instructiunea WAIT?

165 Ce reprezinta BUS LOCK?

166 Lungimea cuvantului extern al microprocesorului se refera la:
a) numarul maxim de biti transferati intre CPU si memorie intr-un ciclu
b) dimensiunea maxima a operandului manipulat direct de ALU
c) latimea magistralei de adrese

167 Lungimea cuvantului intern al microprocesorului se refera la:
a) numarul maxim de biti transferati intre CPU si memorie intr-un ciclu
b) dimensiunea maxima a operandului manipulat direct de ALU

c) latimea magistralei de adrese

168 Cu cat creste lungimea cuvantului extern cu atat:
a) creste viteza de prelucrare a operanzilor
b) creste viteza de obtinerea a operanzilor din memorie
c) creste viteza de executie a instructiunilor

169 Cu cat creste lungimea cuvantului intern cu atat:
a) creste viteza de prelucrare a operanzilor
b) creste viteza de obtinerea a operanzilor din memorie
c) creste viteza de executie a instructiunilor

170 Ce informatii sunt necesare pentru a defini un camp de biti?

171 Valorile logice sunt reprezentate in memorie intr-un:
a) bit
b) octet
c) cuvant

172 Complementul fata de doi se utilizeaza pentru:
a) reprezentarea numerelor naturale
b) reprezentarea numerelor intregi negative
c) reprezentarea numerelor reale negative

173 Complementul fata de unu se utilizeaza pentru:
a) reprezentarea numerelor naturale
b) reprezentarea numerelor intregi negative
c) reprezentarea numerelor reale negative

174 Care este reprezentarea numerelor in complement fata de unu?

175 Care este reprezentarea numerelor in complement fata de doi?

176 Cum sunt reprezentate in memorie caracterele?

177 Ce este sirul de caractere?

178 Cum se specifica lungimea unui sir de caractere?

179 Cum sunt definite inregistrarile ca tip de date?

180 Cum sunt reprezentate numerele reale:
a) in marime si semn
b) in format BCD
c) in format virgula mobila

181 Care este formatul virgula mobila de reprezentare a numerelor reale?

182 Ce formate defineste standardul IEEE 754 pentru reprezentarea numerelor reale?

183 Ce valori speciale sunt definite in standardul IEEE 754?

184 Care sunt elementele de baza care comtribuie la obtinerea oricarui mod de adresare?
a) registre
b) functii de baza
c) obiecte
d) adrese

185 Care sunt obiectele utilizate in modurile de adresare?
a) adunarea
b) deplasarea
c) registre

d) deplasamente
e) adresarea indirecta

186 Care sunt functiile de baza utilizate in modurile de adresare?
a) adunarea
b) deplasarea
c) registre
d) deplasamente
e) adresarea indirecta

187 Care sunt functiile obiectelor de tip registru in modurile de dresare?
a) registru operand
b) registru indirect
c) registru de baza

188 Care sunt functiile campului imediat in modurile de adresare?
a) operand imediat
b) adresa imediata
c) deplasament

189 Care este formula de obtinere a operandului la "adresarea indirecta prin registru"? - daca
g=reg., G(g)=continut.reg., M(x)=adresare indirecta a lui x, a sh b=deplasare a cu b biti la star

190 Care este formula de obtinere a operandului la "adresarea indirecta cu autoincrementare"? - d
g=reg., G(g)=continut.reg., M(x)=adresare indirecta a lui x, a sh b=deplasare a cu b biti la star

191 Care este formula de obtinere a operandului la "adresarea indirecta cu autodecrementare"? -
g=reg., G(g)=continut.reg., M(x)=adresare indirecta a lui x, a sh b=deplasare a cu b biti la star

192 Care este formula de obtinere a operandului la "adresarea bazata cu deplasament"? Daca
g=reg., G(g)=continut.reg., M(x)=adresare indirecta a lui x, a sh b=deplasare a cu b biti la star

193 Care este formula de obtinere a operandului la "adresarea indirecta bazata indexata"? Daca
g=reg., G(g)=continut.reg., M(x)=adresare indirecta a lui x, a sh b=deplasare a cu b biti la star

194 Care este formula de obtinere a operandului la "adresarea indexata indirect bazata "? Daca
g=reg., G(g)=continut.reg., M(x)=adresare indirecta a lui x, a sh b=deplasare a cu b biti la star

195 Care sunt avantajele microprocesoarelor cu un numar mare de registri?
a) lungime mare a programelor in memorie
b) lungime mica a programelor in memorie
c) viteza mare de executie
d) viteza mica de executie

196 La masinile de tip stiva instructiunile opereaza asupra:
a) operanzilor aflati la baza stivei
b) operanzilor aflati in varful stivei
c) operanzilor aflati in memorie

197 Ce procesoare au organizare de tip stiva:
a) CPU de la Intel
b) FPU (coprocesorul aritmetic)
c) transputerele INMOS

198 Registrul de tip acumulator:
a) contine un operand
b) pastreaza rezultatul operatiei aritmetice sau logice

c) pastreaza operatorul

199 Care este functia registrului index?

200 Care este functia registrului de baza?

201 Ce tipuri de intreruperi cunoasteti?

202 La intreruperile vectorizate cererile de intrerupere sunt lansate pe:
a) mai multe linii de intrerupere
b) o linie de intrerupere
c) pe o linie speciala NMI

203 La intreruperile nemascabile cererile de intrerupere sunt lansate pe:
a) mai multe linii de intrerupere
b) o linie de intrerupere
c) pe o linie speciala NMI

204 La intreruperile nevectorizate cererile de intrerupere sunt lansate pe:
a) mai multe linii de intrerupere
b) o linie de intrerupere
c) pe o linie speciala NMI

205 Ce sunt intreruperile cu relansare?

206 Ce este timpul de latenta?

207 Daca mai multe cereri de intrerupere apar simultan microprocesorul:
a) nu trateaza nici o intrerupere
b) accepta cererea cu prioritatea cea mai mare
c) accepta cererea cu prioritatea cea mai mica

208 Ce tipuri de capcane cunoasteti?

209 Cum sunt implementate punctele software de suspendare a programelor?

210 Ce instrumentele se folosesc in depanarea programelor?

211 Care sunt factorii care afecteaza performanta sistemului de calcul?
R: arh.CPU,dimens.si impl set instr.,compilat.cu optim.,tehnolog.cip,SO

212 Performanta calculatorului poate fi descrisa de formula: (ce reprezinta T,p si c?)
a) T*p*c
b) 1/T*p*c
c) T/p*c
R:T=per.ciclu instr.,p=lung.caii,c=ciclii/instr.

213 Care sunt tehnologiile arhitecturale folosite pentru cresterea performantei sistemului de calcul
R: paralelism la nivel instr.(ILP) si ierarhii sofist de mem. Cache

214 Paralelismul la nivelul instructiunilor consta in:
a) executia in paralel a instructiunilor mai multor procese
b) executia acelorasi instructiuni o singura data
c) executia mai multor instructiuni simultan

215 Descrieti implementarea benzii de asamblare (pipeline) a instructiunilor?

216 Ce sunt arhitecturile superscalare?
R: m.m.unit. De calcul puse in paralel

217 Ce este **trace cache**?
R:cache pt instr care pastr instr in ord in care e probabil sa se exec., nu in ord adr

218 La ce se refera **executia speculativa si predictia valorilor**?

R: la dependentele dintre instr. Cand o instr are nevoie de rez. Alteia se ghiceste val rez

219 Memoria cache este introdusa deoarece:
a) memoria RAM este mai rapida ca microprocesorul
b) memoria RAM este mai lenta ca microprocesorul
c) timpi de parcurgere a traseelor mai mari
d) timpi de parcurgere a traseelor mai mici
e) timpi de acces la memorie mai mari
f) timpi de acces la memorie mai mici

220 Memoria cache primara este plasata:
a) pe cipul microprocesorului
b) pe placa de baza
c) pe o placa separata

221 Memoria cache secundara este plasata:
a) pe cipul microprocesorului
b) pe placa de baza
c) pe o placa separata

222 La nivelul memoriei cache primare:
a) se realizeaza o distinctie intre tipurile de date manipulate
b) nu se realizeza o distinctie intre tipurile de date

223 La nivelul memoriei cache secundare:
a) se realizeaza o distinctie intre tipurile de date manipulate
b) nu se realizeza o distinctie intre tipurile de date

224 Implementarea pentru memoria cache pentru cod este:
a) in conducta de executie
b) asociativa

225 Implementarea pentru memoria cache pentru date este:
a) in conducta de executie
b) asociativa

226 Care sunt metodele de reinoire a memoriilor cache?
R: Random Write, FIFO,LRU

227 Care sunt modalitatile de conectare la microprocesor si RAM ale memoriei cache?
R: seriala(look throw), paralela(look aside)

228 Care sunt metodele de depunere in memorie a rezultatelor cand exista cache?
R: Write Through (ce,ci,ram),Write Back(scriere c daca exista, altf ram),Posted Write (ex.buffe

229 Ce tipuri de celule cache cunoasteti?
R:asicr.sincr.,pipelined burst

230 Care sunt clasificarile arhitecturate din pct de vedere al suportului oferit limbajului de nivel inal

231 Arhitectura capabila sa execute programe HLL fara translatare este:
a) arhitectura cu executie directa
b) arhitectura redusa
c) arhitectura orientata catre limbaj
d) arhitectura corelata cu limbajul

232 Arhitectura orientata catre imbunatatirea performantelor la rularea programelor HLL este:
a) arhitectura cu executie directa

b) arhitectura redusa
c) arhitectura orientata catre limbaj
d) arhitectura corelata cu limbajul

233 Arhitectura in care sunt implementate intructiuni specifice si moduri de adresare pentru HLL e
a) arhitectura cu executie directa
b) arhitectura redusa
c) arhitectura orientata catre limbaj
d) arhitectura corelata cu limbajul

234 Arhitectura in care exista o corespondenta biunivoca intre HLL si codul masinii este:
a) arhitectura cu executie directa
b) arhitectura redusa
c) arhitectura orientata catre limbaj
d) arhitectura corelata cu limbajul

235 Care sunt caracteristicile principale ale arhitecturilor RISC?
R: set simpl instr, exec pipeline.nr mare reg cu org in fisier de reg, ierarhia mem

236 Care este coeficientul de multiplicare (ideal) pentru o conducta de executie cu 6 trepte?
a) 3
b) 6
c) 2

237 Organizarea "register window" a registrelor microprocesoarelor cu arhitectura RISC foloseste
a) executia mai multor instructiuni simultan
b) executia instructiunilor intr-un ciclu masina
c) transferul rapid de parametri intre rutine
d) acces mai rapid la memorie

238 Efectele setului redus de instructiuni la arhitecturile RISC sunt:
a) unitate de comanda mica
b) costuri mari de proiectare
c) costuri mici de proiectare
d) unitate simpla de comanda
e) spatiu pentru numar mare de registri

239 Care este aranjamentul "register window" la microprocesoarele cu arhitectura RISC?

240 Care sunt fluxurile care interactioneaza intr-un sistem de calcul?

241 Care este clasificarea lui Flynn?

242 Ce sunt sistemele de calcul de tip SISD?

243 Ce sunt sistemele de calcul de tip SIMD?

244 Ce sunt sistemele de calcul de tip MISD?

245 Ce sunt sistemele de calcul de tip MIMD?

246 Care este clasficarea lui Enslow pentru sistemele MIMD?

247 Care sunt mecanismele fundamentale de care depinde organizarea programelor?
R: mec.czilor, mec datelor

248 Mecanismul comenzilor in organizarea programelor defineste:
a) modul in care un operand este trasferat instructiunii consumatoare
b) modul in care o instructiune produce executia alteia
c) modul in care datele sunt utilizate

249 Mecanismul datelor in organizarea programelor defineste:
a) modul in care un operand este trasferat instructiunii consumatoare
b) modul in care o instructiune produce executia alteia
c) modul in care datele sunt utilizate
250 Care sunt cele 3 categorii de organizare a programelor in functie de momentul executiei instru
R: medii cu derul even impusa de czi, date, cereri
251 Care sunt mediile cu derularea evenimentelor impusa de comenzi?
R:logica de cda determ ord de executie, programe secventiale, ex contor hard
252 Care sunt mediile cu derularea evenimentelor impusa de date?
R: o instr se exec cand toti operanzi sunt dispon
253 Care sunt mediile cu derularea evenimentelor impusa de comenzi?
R: o instr se exec cand toti operanzi sunt dispon si rezult e solicit
254 Care sunt cele 3 tipuri de transfer de date intre instructiunea consumatoare si cea producatoa
R: trasf prin refe, transf prin val, transf prin nume simbolic
255 Cum sunt reprezentate programele la masinile bazate pe flux de date?
a) prin organigrame care indica fluxul de comenzi
b) prin grafuri orientate ce indica interdependenta datelor
c) secvente de instructiuni
256 Mecanismul comenzilor in masinile "data flow" este de tip:
a) derulare impusa de comenzi
b) derulare impusa de date
c) derulare impusa la cerere
257 Transferul datelor la masinile "data flow" este prin:
a) referire
b) valoare
c) nume simbolic
258 Mecanismul comenzilor in masinile bazate pe flux de comenzi este de tip:
a) derulare impusa de comenzi
b) derulare impusa de date
c) derulare impusa la cerere
259 Transferul datelor la masinile bazate pe flux de date este prin:
a) referire
b) valoare
c) nume simbolic
260 La masinile "data flow" (bazate pe flux de date) sincronizarea:
a) trebuie specificata in mod explicit
b) se face automat
c) uneori trebuie specificata, altori este automata
261 Care sunt unitatile componente ale unui calculator "data flow"?
R:unit de combinare, ALU, sir de instr, sir de rez
262 Unitatea de combinare de la masina "data flow" are rol de:
a) unitate de comanda
b) unitate de decodificare
c) unitate aritmetica si logica

263 Ce este o retea de calculatoare?
R:colect de calc autonome intercon prin canale de c, fara rel master-slave
264 Care sunt obiectivele urmarite in realizarea retelelor de calculatoare?
R: disponibilit resurselor, fiabilit,economie,scalabilitate, mediu puternic de comunic
265 Structura retelei de calculatoare se refera la:
a) organizarea abstracta pe mai multe niveluri ierarhice
b) caracteristicile tehnice ale retelei
266 Care este structura unei retele de calculatoare?
R: hosturi interconectate prin subretele
267 O subretea este formata din:
a) hosturi
b) elemente de comutare
c) linii de transmisie
268 Ce tipuri de subretele de comunicatie cunoasteti?
R: pct la pct., de difuzare
269 In subretelele cu difuzare exista:
a) mai multe canale de comunicatie
b) un singur canal de comunicatie
c) nici un canal de comunicatie
270 Cum se realizeza partajarea canalului la subretelele cu difuzare?
a) prin alocare statica
b) prin alocare dinamica
271 Alocarea statica a canalului in subretelele cu difuzare se realizeaza prin:
a) folosirea unei unitati de arbitrare a canalului
b) multiplexare in frecventa
c) multiplexare in timp
272 Alocarea dinamica a canalului in subretelele cu difuzare se realizeaza prin:
a) folosirea unei unitati de arbitrare a canalului
b) multiplexare in frecventa
c) multiplexare in timp
273 Scopul fiecarui nivel in arhitectura retelelor este de a:
a) comunica cu celelalte niveluri
b) de a oferi servicii nivelului superior
c) furniza anumite informatii utilizatirului
274 Protocoalele implementate intre niveluri corespondente implementeaza:
a) o comunicatie fizica intre acele niveluri
b) o comunicatie virtuala intre acele niveluri
c) transmiterea datelor de la un nivel ierarhic la altul
d) o comunicatie intre doua entitati pereche
275 Care sunt nivelurile modelului OSI pentru retele de calculatoare?
276 Nivelul "fizic" al modelului OSI pentru retele de calculatoare se ocupa cu:
a) transmiterea datelor fara erori nivelului retea
b) controlul operarii subretelei
c) transmiterea sirurilor liniare de biti prin canalul de comunicatie

277 Nivelul "legaturii de date" al modelului OSI pentru retele de calculatoare se ocupa cu:
a) transmiterea datelor fara erori nivelului retea
b) controlul operarii subretelei
c) transmiterea sirurilor liniare de biti prin canalul de comunicatie

278 Nivelul "retea" al modelului OSI pentru retele de calculatoare se ocupa cu:
a) transmiterea datelor fara erori nivelului retea
b) controlul operarii subretelei
c) transmiterea sirurilor liniare de biti prin canalul de comunicatie

279 Primul nivel al modelului OSI care realizeza o comunicatie end-to-end intre sursa si destinatie
a) nivelul retea
b) nivelul sesiune
c) nivelul prezentare
d) nivelul transport

280 Serviciul orientat pe conexiune in retelele de calculatoare este:
a) similar serviciului postal
b) similar serviciului telefonic
c) nu este similar nici cu cel postal, nici cu cel telefonic

281 Serviciul neorientat pe conexiune in retelele de calculatoare este:
a) similar serviciului postal
b) similar serviciului telefonic
c) nu este similar nici cu cel postal, nici cu cel telefonic

282 Sistemele multiprocesor cu structura matriciala se incadreaza in clasa sistemelor:
a) SISD
b) SIMD
c) MISD
d) MIMD

283 Sistemele multiprocesor vectoriale (structura pipeline) se incadreaza in clasa sistemelor:
a) SISD
b) SIMD
c) MISD
d) MIMD

284 Metoda bitului de paritate este o metoda prin care se face:
a) detectia erorilor simple
b) corectia erorilor simple
c) detectia erorilor duble
d) reconfigurarea memoriei

285 Codul Hamming este o metoda prin care se face:
a) detectia erorilor simple
b) corectia erorilor simple
c) detectia erorilor duble

d) reconfigurarea memoriei

286 Tehnica "bitilor de rezerva" este utilizata pentru
a) detectia erorilor simple
b) corectia erorilor simple
c) detectia erorilor duble
d) reconfigurarea memoriei

287 Transputerele INMOS au:
a) numar mare de registri
b) numar mic de registri
c) numar mediu de registri

288 Care sunt unitatile functionale ale transputerelor?

289 Planificatorul microcodat al transputerului este utilizat pentru:
a) implementarea comunicatiei
b) implementarea mecanismului de concurenta
c) interfatarea cu memoria

290 Arhitectura CPU a transputerului este:
a) CISC
b) RISC

291 Descrieti setul de registri CPU ai transputerului.

292 Descrieti setul de registri FPU ai transputerului.

293 Care sunt instructiunile speciale din setul de instructiuni al transputerului?

294 Formatul instructiunilor transputerului:
a) este unic
b) are doua campuri
c) este diferit pentru tipuri de instructiuni

295 Functia "operate" la transputer faciliteaza:
a) extinderea operanzilor instructiunilor
b) interpretarea registrului operand ca operatie (cod instructiune)
c) operarea cu stiva de evaluare

296 Functia "prefix" la transputer faciliteaza:
a) extinderea operanzilor instructiunilor
b) interpretarea registrului operand ca operatie (cod instructiune)
c) operarea cu stiva de evaluare

297 Un canal de comunicatie la transputer se implementeaza
a) printr-un cuvant de memorie
b) legatura de punct la punct
c) legatura multipunct

Alte întrebări curs

1. Generații de calculatoare
2. Structura sistemului de calcul
3. Memoria sistemului de calcul - caracteristici.
4. Ierarhia sistemului de memorie
5. Memoria externă a sistemului de calcul
6. Microprocesorul – descriere generală
7. Unitatea centrală de prelucrare
8. Dispozitive de I/E
9. Comunicația în sistemul de calcul
10. Rețele de calculatoare. Caracteristici generale.
11. Structura retelelor de calculatoare
12. Arhitectura rețelelor de calculatoare.
13. Modelul OSI pentru rețele de calculatoare.
14. Programare Internet. Arhitecturi software.
15. Identificarea resurselor Internet (URL)
16. Limbajul HTML.
17. Definiția sistemului calcul.
18. Care sunt elementele componente ale unui sistem de calcul?
19. Ce reprezintă arhitectura unui sistem de calcul?
20. Specificați tipurile de decompoziții în arhitectura sistemelor de calcul.

21. Specificați care este principiul care stă la baza arhitecturii multistrat.
22. Arhitectura multistrat a sistemului de calcul.
23. Ce este masina virtuala? Cum se numeste limbajul de programare al acesteia?
24. Care sunt metodele de executare a programelor în arhitectura multistrat?
25. Ce este translatarea?
26. Ce este interpretarea?
27. Exista o granita clara intre hardware si software referitor la implementarea unor anumite functii? Justificati raspunsul.
28. Ce componente cuprinde nivelul microarhitecturii?
29. Ce componente cuprinde nivelul logic digital?
30. Ce reprezintă nivelul ISA in arhitectura multistrat?
31. Ce este microprocesorul?
32. Care sunt unitățile funcționale componente ale microprocesorului?
33. Ce este UAL?
34. Ce este unitatea de comandă?
35. Ce sunt regiştrii microprocesorului?
36. Ce sunt semnalele de comandă?
37. Ce sunt semnalele de stare?
38. Ce reprezintă numărul de biți al microprocesorului?
39. Ce reprezintă magistrala de adrese?
40. Ce reprezintă magistrala de date?
41. Ce reprezintă registrul contor de instrucțiuni?
42. Ce reprezintă registrul „Buffer de adrese”?

43. Ce reprezintă registrul „Buffer de date”?
44. Ce reprezintă registrul de instrucțiuni?
45. Ce reprezintă decodificatorul de instrucțiuni?
46. Ce reprezintă registrul de stare (indicatorilor de condiții)?
47. **Descrieți funcționarea UCP a microprocesorului.**
48. Care este organizarea memoriei la microprocesorul I8086?
49. Specificați modalitatea de obținere a datelor din memorie, utilizând registrii microprocesorului.
50. Care sunt regiştrii segment ai microprocesorului I8086?
51. Care sunt regiştrii offset ai microprocesorului I8086?
52. Care sunt regiştrii cu scop general ai microprocesorului I8086?
53. Care sunt regiştrii utilizați în adresarea segmentului de cod?
54. Care sunt regiştrii utilizați în adresarea segmentului de date?
55. Care sunt regiştrii utilizați în adresarea segmentului de stivă?
56. Care sunt regiştrii utilizați în adresarea extra segmentului de date?
57. Ce reprezintă indicatorii de condiție (flag-urile) din registrul de stare?
58. Ce reprezintă flagul „Carry”?
59. Ce reprezintă flagul „Auxiliary carry”?
60. Ce reprezintă flagul „Parity”?
61. Ce reprezintă flagul „Zero”?
62. Ce reprezintă flagul „Sign”?
63. Ce reprezintă flagul „Overflow”?
64. Ce reprezintă flagul „Direction”?

65. Ce reprezintă flagul „Interrupt”?
66. Ce reprezintă flagul „Trap”?
67. Ce reprezintă un ciclu instrucțiune?
68. Ce cicluri maşină cunoşteți?
69. Care este mnemonica generală a instrucțiunilor I8086?
70. Ce pot fi operanzii destinație din instrucțiuni?
71. Ce pot fi operanzii sursă din instrucțiuni?
72. Ce tipuri de instrucțiuni de transfer cunoaşteți?
73. Care sunt instrucțiunile cu stiva?
74. Care sunt instrucțiunile cu registrul acumulator?
75. Care sunt instrucțiunile cu obiect adresă?
76. Care sunt instrucțiunile pentru transferul flag-urilor?
77. Ce reprezintă BCD?
78. Care sunt instrucțiunile de adunare?
79. Care sunt instrucțiunile de scădere?
80. Care sunt instrucțiunile de Inmulțire?
81. Care sunt instrucțiunile de împărțire?
82. Ce reprezintă corecția zecimală la adunare?
83. Ce reprezintă corecția ASCII la adunare?
84. Ce reprezintă corecția zecimală la scădere?
85. Ce reprezintă corecția ASCII la scădere?
86. Care sunt instrucțiunile de deplasare?
87. Ce reprezintă deplasarea aritmetică?

88. Care sunt instrucțiunile de rotire?

89. Care sunt instrucțiunile logice diadice?

90. Care sunt instrucțiunile primitive cu şiruri?

91. Ce reprezintă prefixele asociate instrucțiunilor cu şiruri?

92. Care sunt instrucțiunile de transfer necondițional al controlului programului?

93. Care sunt instrucțiunile de transfer condițional al controlului programului?

94. Care sunt instrucțiunile pentru controlul iterațiilor?

95. Ce reprezintă întreruperile?

96. Care este mecanismul de execuție a întreruperilor program?

97. Cum se calculează adresa rutinei de întrerupere?

98. Care sunt instrucțiunile de setare a indicatorilor de condiție?

99. Care sunt instrucțiunile referitoare la starea procesorului?

100. Ce reprezintă memoria virtuală?

101. Cum se calculează adresa efectivă de memorie la I8086?

102. Codificarea instrucțiunilor (formatul general al instrucțiunilor).

103. Ce reprezintă modurile de adresare?

104. Care sunt elementele de bază pentru obținerea modurilor de adresare?

105. Care sunt obiectele utilizate în modurile de adresare?

106. Specificați adresarea imediată.

107. Specificați adresarea directă prin registru.

108. Specificați adresarea indirectă prin registru.

109. Specificați adresarea indirectă cu autoincrementare/autodecrementare..

110. Specificați adresarea indirectă bazată cu deplasament.

111. Specificați adresarea dublu indirectă bazată cu deplasament.

112. Specificați adresarea indirectă bazată indexată.

113. Specificați adresarea indirectă indirect bazată cu deplasament.

114. Specificați adresarea indirectă cu deplasament indirect bazată indexată cu deplasament.

115. Care sunt factorii care afectează performanța sistemului de calcul?

116. Formula de calcul a performanței microprocesorului.

117. Care sunt tehnologiile arhitecturale de creştere a performanței microprocesorului?

118. Paralelismul instrucțiunilor.

119. Descrieți funcționarea conductei de execuție a instrucțiunilor.

120. Ce sunt arhitecturile superscalare?

121. Ce reprezintă hazardul conductei de execuție a instrucțiunilor?

122. Ce tipuri de hazard cunoaşteți?

123. Ce este memoria cache?

124. Tipuri de memorie cache.

125. Care sunt caracteristicile arhitecturilor RISC?

126. Care sunt avantajele arhitecturilor RISC?

127. Care este organizarea „register window” la RISC?

128. Ce avantaje aduce setul mic de instrucțiuni la RISC?

129. Ce avantaje aduce numarul mare de regiştri la RISC?

130. Specificați ce reprezintă întreruperile hardware.

131. Tipuri de întreruperi hardware.

132. Intreruperi vectorizate.

133. Întreruperi nemascabile

134. Întreruperi nevectorizate.

135. Întreruperi cu relansare.

136. Ce reprezintă timpul de latentă.

137. Microarhitectura UltraSPARCII.

138. Ce reprezintă un sistem multiprocesor?

139. Care sunt formele extreme de paralelism?

140. Care sunt fluxurile care interacționează într-un sistem de calcul (Flynn)?

141. Clasificarea lui Flynn.

142. Clasificarea lui Enslow.

143. Tipuri de arhitecturi de sisteme de calcul de tip paralel.

144. Unități funcționale multiple specializate.

145. Procesoare asociative.

146. Calculatoare orientate pe flux de date.

147. Schema bloc a unui calculator orientat pe flux de date.

148. Care sunt funcțiile unității de combinare la calculatoarele orientate pe flux de date.

149. Calculatorul orientat pe flux de date Multi-ALU.

150. Calculatorul orientat pe flux de date Multi-ring.

151. Care este contextul actual al sistemelor de calcul?

152. Arhitectura client-server.