Corneliu BURILEANU

BAZELE ARHITECTURII MICROPROCESOARELOR CISC și RISC

Editura Academiei Oamenilor de Știință din România

București, 2025

Descrierea CIP a Bibliotecii Naționale a României
BURILEANU, CORNELIU
Bazele arhitecturii microprocesoarelor CISC și RISC /
Corneliu Burileanu. - București :
Editura Academiei Oamenilor de Știință din România, 2025

ISBN 978-630-6518-57-9

004

CUPRINS

Partea a II-a: Atribute evoluate de arhitectură

Partea a III-a: Alte exemple de arhitecturi

PREFAȚĂ

Scurtă istorie subiectivă

După părerea autorului, momentul de cotitură în dezvoltarea tehnicii de calcul și de aici toate aplicațiile spectaculoase care au urmat (de la telecomunicații, Internet, la inteligența artificială) a fost apariția **tranzistorului**. Dispozitivul electronic care a schimbat istoria a fost inventat la Bell Telephone Laboratories din New Jersey de *John Bardeen*, *Walter Houser Brattain*, și *William Bradford Shockley* și anunțat pe 6 decembrie 1947.

Până la apariția primelor **circuite integrate** în care au fost interconectate pe același suport de siliciu mai multe tranzistoare nu a mai fost decât un pas. Din 1960 dezvoltarea circuitelor integrate a permis apariția generației a 3-a de calculatoare (IBM 360, GE 635).

În aprilie 1971, compania Intel anunță apariția unui nou dispozitiv: **microprocesorul**. Primul prototip a fost numit **4004**, un procesor pe 4 biți. La scurt timp, în aprilie 1972, Intel anunță microprocesorul cu numele de cod **8008** care poate fi considerat prototipul generației I de microprocesoare pe care acum le numim **CISC** (***Complex Instruction Set Computer***).

Din acel moment microprocesoarele s-au dezvoltat continuu. În aprilie 1974 Intel anunță microprocesorul pe 8 biți **8080**, prototip pentru generația a II-a de microprocesoare și pentru conceptul CISC. În 1976, un grup de ingineri plecați de la Intel și care fondaseră o nouă companie, Zilog, anunță apariția unei variante mult îmbunătățită a procesorului 8080: **Z80**. Intel răspunde în același an cu lansarea unui nou tip de dispozitiv care este un microcalculator pe un singur cip (tot pe 8 biți): **8085**.

În 1977 compania Texas Instruments lansează prima tentativă de a impune un microprocesor pe 16 biți: **TI 9900**. Această încercare nu a avut succes pe piață. Abia în iunie 1978 Intel lansează microprocesorul pe 16 biți **8086** care a devenit prototip pentru generația a III-a de microprocesoare și care este o adevărată emblemă a familiilor de microprocesoare CISC care au urmat. Ulterior, alte companii au anunțat competitori ai lui 8086: **Z8000** de la Zilog (din păcate sub așteptări) și **M68000** de la Motorola (un procesor cu performanțe superioare lui 8086 dar fără acea susținere cu sisteme de dezvoltare

și fără promovarea agresivă a procesoarelor Intel). Mai menționăm un moment care a fost mai puțin mediatizat: în iunie 1979 Intel lansează microprocesorul **8088**, pe 16 biți, dar având magistralele de legătură cu alte dispozitive sau cu echipamentele periferice pe 8 biți.

În 1981 compania IBM a anunțat echiparea calculatoarelor sale PC-XT cu procesoare Intel 8086 și 8088. În februarie 1982 Intel lansează microprocesorul **80286**, tot pe 16 biți dar primul care poate fi folosit în **modul virtual**. El devine prototip pentru generația a IV-a de microprocesoare. Ca urmare, în 1984 IBM dotează calculatoarele PC-AT (Advanced Technology) cu Intel 80286. După apariția microprocesoarelor pe 32 biți **Intel i386 DX** (în 1985), și **Intel i486 DX** (în 1989), în martie 1993 apare **Intel Pentium** prototip pentru generația a V-a de microprocesoare (tot 32 biți). Urmează Intel **Pentium Pro** (1995), **Pentium II** (1997), **Pentium III** (1999), **Pentium 4** (2000), procesoarele **Core** (de pildă, **Intel Core2 Duo** în 2006).

Între timp, în 1980 apar primele microprocesoare **RISC** (***Reduced Instruction Set Computer***) dezvoltate la universitățile Berkeley (**RISC I** și **RISC II)** și Stanford (**MIPS**). Caracteristicile acestei clase de procesoare au devenit repede foarte atractive pentru o serie de aplicații și au fost preluate de mai multe companii: **Sparc** (Sun Microelectronics), **MIPS** (Mips Technologies), **PowerPC** (Apple, IBM, Motorola), **PA-RISC** (Hewlett-Packard), **Alpha** (Digital Semiconductor).

Compania Acorn Computers Limited din Cambridge, Anglia, este cea care a lansat familia de procesoare **ARM** (se va citi ca un cuvânt și nu ca un șir de inițiale). Pe 26 aprilie 1985 a apărut primul membru al familiei: ARM1 cu 25.000 tranzistoare, tehnologie CMOS 3 microni. Numele este un acronim de la „**A**corn **R**ISC **M**achine". Dezvoltarea procesorului a fost preluată în 1990 de un consorțiu numit Advanced Risc Machines Ltd. în care inițial erau Apple Computer și VLSI Technology Inc. Procesorul a fost dezvoltat în continuare sub același nume care acum este acronimul pentru „**A**dvanced **R**ISC **M**achine". Caracteristicile familiei ARM sunt atât de interesante încât a ajuns să fie unul dintre principalele microprocesoare pe piață cu un avans considerabil în utilizarea lor pentru dispozitivele mobile. Avantajele procesoarelor ARM țin de simplitatea arhitecturii și structurii, consum redus de putere, versatilitate și faptul că diverse companii au putut cumpăra licența și fabrica propriile variante. Câteva exemple de procesoare ARM:

- proiectate de consorțiul ARM – **ARM10**, **Cortex**, **Neoverse**;

- proiectate de alte companii – **StrongARM** (Digital), **Faraday** (Faraday Technology), **XScale** (Intel), **Snapdragon** (Qualcomm), **Denver** și **Carmel** (Nvidia), **Exynos** (Samsung) și altele.

Conceptul RISC lansat la universitățile Berkeley și Stanford a permis continuarea dezvoltării unor microprocesoare pe 32 biți performante. După **RISC I** și **II**, au apărut **RISC III** (în 1984) și **RISC IV** (în 1988). Pornind de la ideea că arhitecturile existente erau supuse restricționărilor ce decurg din patente și licențe obligatorii, a fost dezvoltată în 2010 o **arhitectură deschisă**, folosibilă fără restricții: **RISC V** (se va citi *risc five*) care a fost făcută publică în 2011. În 2015 a fost creată „RISC V Foundation" (devenită apoi „RISC V International") menită să gestioneze o platformă colaborativă pentru dezvoltări ulterioare ale microprocesorului.

Am început scurta incursiune în istoria microprocesorului cu apariția tranzistorului 1947. Recent, în luna martie 2024, compania Cerebras a anunțat fabricarea celui mai mare cip obținut până în acel moment, **WSE-3**, cu aria 46,255 mm^2 (cam 75% din dimensiunea unei coli de hârtie format A4), conținând 4 x 10^{12} tranzistoare grupate în 900.000 de nuclee de procesoare, cele mai multe specializate pentru prelucrări specifice inteligenței artificiale. În continuare ne așteptăm ca siliciul să fie înlocuit cu **grafene** (o formă cristalină de carbon) care permite realizarea unor dispozitive cu viteze superioare celor fabricate pe siliciu. Ne așteptăm să continue dezvoltarea unei noi clase de dispozitive electronice „**memristorul**" cu caracteristici asemănătoare sinapselor din creierul uman și care permite realizări în ceea ce se numește „arhitectura neuromorfică".

Despre volumul de față

Lucrarea se adresează în primul rând studenților Facultăților de Electronică, Telecomunicații și Tehnologia Informației. Trebuie precizat însă că este necesar ca cititorul trebuie să fie familiarizat cu circuitele digitale, cel puțin la nivel de blocuri funcționale. De asemenea este necesară cunoașterea structurilor de date și tehnicilor de programare în limbaje de nivel înalt. Materialul prezentat aici a fost predat timp de patru decenii studenților din Politehnica București (desigur, cu actualizări continue) și reflectă experiența autorului în proiectarea, implementarea și utilizarea multor familii de microprocesoare [6], [7], [8], [10]. Sperăm ca lucrarea să fie utilă și unei categorii de ingineri specialiști în hardware și software care pot găsi o abordare generală sistematică a atributelor de arhitectură ale microprocesoarelor CISC și RISC dar și date concrete despre trei familii reprezentative: x86, ARM și RISC V.

Lucrarea se bazează pe trei concepte fundamentale pentru abordarea domeniului: **arhitectură**, **semnificație**, **funcție**.

Cititorul trebuie să aibă în vedere modul în care aceste concepte ghidează înțelegerea explicațiilor:

- **Arhitectura** unui sistem nu este același lucru cu structura sa. Atributele de arhitectură ale microprocesoarelor vor fi explicate detaliat cu exemple pentru familiile de procesoare cele mai utilizate. Nu neglijăm însă modul în care arhitectura impune anumite caracteristici de structură și, respectiv, modul în care structura limitează tipul și flexibilitatea atributelor de arhitectură.
- Șirurile de cifre binare care sunt procesate sau sunt stocate în memorie și/sau în porturi contează doar în măsura în care le atașăm o **semnificație**. Numai astfel putem discerne între operanzi/rezultate, coduri sau adrese. O zonă de stocarea informației este considerată „goală” doar dacă cifrelor binare aflate acolo nu li se atașează nicio semnificație.
- Circuitele care constituie suportul fizic al oricărui procesor pot fi organizate sub formă de blocuri funcționale. Așadar, ne interesează **funcția** realizată de fiecare bloc. Detaliile de structură nu sunt întotdeauna relevante pentru înțelegerea funcționării procesorului și familiarizarea cu atributele sale de arhitectură.

Volumul își propune să fie o pledoarie pentru motivarea studenților de a acorda atenție aprofundării cunoștințelor despre dispozitivul electronic numit microprocesor și însușirii deprinderilor de a utiliza diverse limbaje de asamblare. Este evident că un inginer care primește sarcina de a dezvolta rapid o aplicație folosind un sistem de calcul concret alege folosirea unui limbaj de nivel înalt care îi permite rezolvări rapide și cu costuri minime. Varianta utilizării detaliilor de arhitectură ale unui anumit microprocesor și dezvoltarea unor aplicații în limbaj de asamblare va avea însă, întotdeauna, avantajul unei optimizări a modului de rezolvare a sarcinii impuse. În plus, ne așteptăm ca studenții facultăților de Electronică, viitori ingineri electroniști, să aibă contribuții hotărâtoare în proiectarea unor noi procesoare sau în dezvoltarea unora existente. O serie de lucrări pledează pentru importanța cunoașterii atributelor de arhitectură ale microprocesoarelor și a limbajelor de asamblare [1], [2], [3], [4], [5], [9].

Structura volumului este următoarea:

- Capitolul 1 introduce definirea noțiunilor de bază folosite ulterior și trece în revistă câteva dintre convențiile de reprezentare a informației în sistemele digitale.
- Capitolul 2 prezintă schemele bloc funcționale ale principalelor componente ale unui nucleu de procesor CISC.
- Capitolele 3, 4, 5, 6 și 7 detaliază principalele caracteristici ale atributelor de arhitectură: registre, organizarea memoriei, tehnici de adresare, setul de instrucțiuni, strategii de intrare/ieșire. Încercăm o sistematizare a caracteristicilor, în general, dar și precizări pentru cele trei familii de procesoare vizate: x86, ARM și RISC V.
- Capitolele 8 și 9 se ocupă de condiționările reciproce între arhitectură și structură pentru procesoarele CISC, respectiv RISC.
- Capitolul 10 prezintă particularitățile desfășurării în timp a instrucțiunilor CISC și RISC insistând asupra abaterilor de la secvențele normale.
- Capitolele 11, 12 și 13 detaliază atribute evoluate de arhitectură: gestionarea memoriei virtuale, mecanismul protecției resurselor sistemului de calcul, multiprocesarea, cu descrierea principiilor și a cazurilor concrete pentru cele trei familii de procesoare.
- Capitolele 14, 15 și 16 prezintă arhitecturi alternative: coprocesoarele aritmetice, o familie de microcontrolere și caracteristicele procesoarelor grafice (GPU).

Subliniem câteva particularități ale prezentărilor:

- Parte dintre figuri prezintă **scheme bloc funcționale**, simplificate, cu un cod al culorilor care să permită o înțelegere mai ușoară a prelucrării informației.
- Punem un accent deosebit pe **descrierea formală a semanticii** instrucțiunilor și, în general a acțiunilor microprocesorului, ca o alternativă elegantă la explicațiile în cuvinte.
- Fiecare capitol are indicate **propriile referințe listate în ordine alfabetică**. La finalul volumului sunt indicate și o serie de referințe suplimentare.

Îmi fac o plăcută datorie de a exprima mulțumirile și recunoștința echipelor de tineri colaboratori care m-au însoțit în toți acești ani în abordarea domeniului și Editurii Academiei Oamenilor de Știință care a acceptat să editeze și să publice volumul.

REFERINȚE

[1] Agarwal, K. K. and A. Agarwal, "Do we need a separate assembly language programming course?", The Journal of Computing in Colleges. Vol. 19, No. 4, pp. 246-251 (2004).

[2] Anguita M. and F. J. Fernadez-Baldomer, "Software optimization for improving students motivation in a computer architecture course", IEEE Transactions on Education, Vol. 50, No. 4, pp. 373-378 (2007).

[3] Bolanakis, D. E. , G. A. Evangelakis, E. Glavas and K. T. Kotsis, "A Teaching Approach for Bridging the Gap between Low-level and Higher-level Programming using Assembly Language Learning for Small Microcontrollers", Computer Application in Engineering Education, Vol. 19, Issue 3, pp. 525-537 (2011).

[4] Bolanakis, D. E.. E. Glavas, and G. A. Evangelakis, "An integrated microcontroller based tutoring for computer architecture laboratory course", International Journal of Engineering Education. Vol. 23, No. 4, pp. 785-798 (2007).

[5] Buckner, K., "A non-traditional approach to an assembly language course", The Journal of Computing in Colleges. Vol. 22, No. 1, pp. 179-186 (2006).

[6] Burileanu C. et al., „Arhitectura microprocesoarelor – Îndrumar de laborator" Politehnica București, 1994, 1997, 1998

[7] Burileanu, C., „Arhitectura microprocesoarelor", Editura Denix, 1994, ISBN 973-95811-1-0

[8] Burileanu, C., Mihaela Ioniță, M. Ioniță, M. Filotti, „Microprocesoarele x86 – o abordare software", Editura "Grupul Microinformatica", Cluj-Napoca, 1999, ISBN 973-9215-95-5.

[9] Ditcher, A. K., "Effective teaching and learning in higher education, with particular reference to the undergraduate education of professional engineers", International Journal of Engineering Education. Vol. 17, No. 4, pp. 24-29 (2001).

[10] Șandru, Elena-Diana, G. V. Popescu, H. Cucu, C. Burileanu, „Microprocessor Architecture. Laboratory Guide", Politehnica București, MatrixRom, 2020, ISBN 978-606-25-0548-6.

1

STRUCTURA UNUI CALCULATOR. DEFINIȚII

1.1 COMPONENTELE FUNCȚIONALE ALE UNUI CALCULATOR BAZAT PE PRINCIPIILE „VON NEUMANN”

Definiția microprocesorului va fi dată într-un mod indirect; vom defini mai întâi sistemul din care acesta face parte (calculatorul) pentru a-l putea delimita funcțional în cadrul sistemului.

Calculatorul, structurat ca o mașină **VON NEUMANN**, este un sistem programabil de prelucrarea informației care are două componente inseparabile și definitorii: **hardware** și **software** [1].

A. Din punct de vedere **hardware**, calculatorul are trei componente funcționale conectate într-un mod specific (Figura 1.1) [7], [10]:

1. UNITATEA CENTRALĂ DE PRELUCRARE – UCP (***C****entral* ***P****rocessing* ***U****nit* – **CPU)** are două funcții esențiale:

- prelucrarea datelor;
- controlul activității întregului calculator.

O **Unitate centrală de prelucrarea informației** având funcțiile enunțate mai sus, care coordonează un sistem structurat funcțional ca în Figura 1.1 și care, fizic, se prezintă sub forma unui singur cip, se numește **MICROPROCESOR (µP)**. Pe parcursul acestei lucrări îl vom numi și simplu, **PROCESOR**.

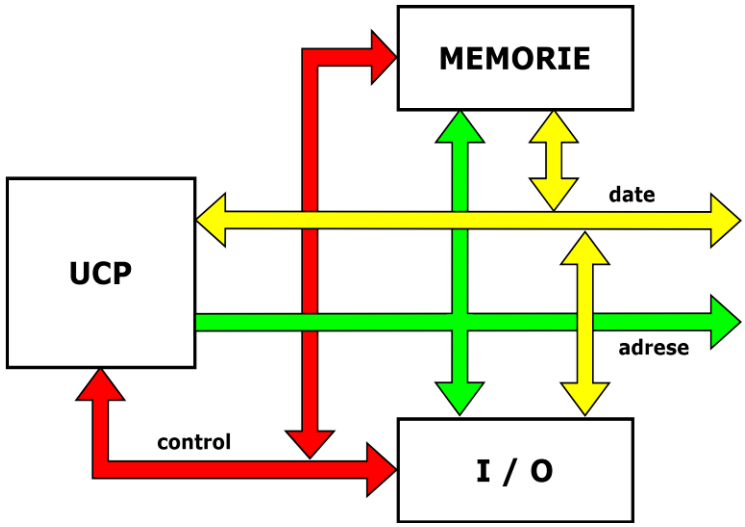

Fig. 1.1 Blocurile funcționale ale unui calculator

2. MEMORIA este, din punctul de vedere al sistemului pe care îl definim, o secvență de locații pentru stocarea informației. Fiecare locație este definită prin două entități informaționale:

• **Conținutul** reprezintă de o înșiruire de cifre binare **0** sau **1** (**biți**); nu am folosit noțiunea de „număr binar", pentru că informația stocată într-o locație de memorie poate avea diverse semnificații:

- numere
- coduri etc.

Numărul de cifre binare conținute într-o locație depinde de modul în care microprocesorul organizează informația în memorie; mărimea unei locații va fi denumită **formatul memoriei**, exprimat în număr de biți (de regulă 8, 16, 32 sau 64 biți). Insistăm asupra faptului că, din punctul nostru de vedere, formatul memoriei nu are nicio legătură cu organizarea fizică a cipurilor de memorie.

• **Adresa** reprezintă numărul de ordine al locației, care permite identificarea sa în cadrul secvenței de locații (există o corespondență biunivocă între fiecare locație de memorie și adresa sa).

Referindu-ne la memoria unui calculator vom folosi câteva noțiuni aferente:

- **Harta memoriei**: totalitatea locațiilor de memorie pe care le poate adresa un microprocesor.
- **Pagini** și/sau **segmente**: subdiviziuni *logice* ale hărții memoriei, ale căror dimensiuni, fixe sau dinamice, sunt specifice modului în care un anumit microprocesor organizează memoria.

Subliniem din nou că aceste moduri de organizare nu au nici-o legătură cu structura fizică a memoriei unui calculator care este formată din unul sau mai multe cipuri cu capacități diverse. Capacitatea totală de stocare a informației pe care o realizează fizic cipurile de memorie într-un calculator este definită ca *memorie internă*. Aceasta nu acoperă, în mod necesar, harta memoriei aferentă microprocesorului respectiv.

În ceea ce privește semnificația conținutului memoriei calculatorului, aceasta delimitează în harta memoriei două zone (fizic această delimitare este aproape total transparentă):

- **memoria de date** conține operanzi și/sau rezultate; fizic, această porțiune de memorie este de tip RAM (memorie cu scriere/citire).
- **memoria de program** care conține instrucțiuni; de regulă, (dar nu obligatoriu) această zonă este o memorie de tip ROM (memorie din care se poate doar citi).

Dacă în ceea ce privește memoria de date semnificația este limpede și nu trebuie insistat, pentru memoria de program apare o noțiune nouă: **instrucțiunea**. Vom înțelege prin **instrucțiune** informația codificată (binar) prin care se impune microprocesorului desfășurarea unei acțiuni specifice. Definiția trebuie explicată cu următoarele observații:

- Fiecare instrucțiune este asociată în mod biunivoc cu un șir de cifre binare; deoarece acestea „codifică” instrucțiunile, vor fi denumite **coduri**.
- O instrucțiune reprezintă cea mai simplă acțiune, cu rezultat bine precizat, din activitatea unui calculator a cărui *unitate centrală de prelucrare a informației* este un microprocesor anume.
- Un microprocesor concret poate „recunoaște” și executa numai codurile corespunzătoare instrucțiunilor pentru care a fost construit; totalitatea instrucțiunilor pe care un microprocesor le poate recunoaște și executa alcătuiește **setul de instrucțiuni** al microprocesorului respectiv.

- Înșiruirea instrucțiunilor în memoria de program nu este haotică ci sub formă de **programe**, noțiune definită ca fiind o secvență de coduri de instrucțiuni organizate în mod logic și coerent după un anumit algoritm, astfel încât întregul calculator să execute o sarcină prestabilită. Noțiunea de **sarcină** (**task**) nu trebuie confundată cu cea de program: sarcina unui calculator corespunde unei alocări dinamice a resurselor hardware și software; există sarcini pentru a căror îndeplinire sunt necesare mai multe programe.

Se poate conchide că semnificația conținutului locațiilor de memorie este conferită de programator în concordanță cu funcțiile specifice realizate de microprocesor: putem vorbi de *numere binare* atunci când ne referim la date (operanzi/rezultate) dar vom folosi noțiunea de *coduri* când ne referim la instrucțiuni sau reprezentarea altor tipuri de informații. Este foarte important de remarcat că în schema bloc funcțională propusă (Figura 1.1) **memoria nu are nici un control asupra semnificației informației pe care o conține**.

3. DISPOZITIVELE DE INTRARE/IEȘIRE (I/O) sunt constituite din circuitele prin care se realizează legătura între calculator și lumea exterioară. O unitate elementară de conversație cu exteriorul poartă numele de **port de intrare/ieșire**.

Între porturi și locațiile din memorie există niște similitudini care determină pe unii autori să le asimileze într-un bloc funcțional unic:

- Porturile sunt în esență tot locații de memorare a informației, adresabile; informația care se folosește uzual aici este alcătuită din operanzi/rezultate (date).
- Există, în mod similar, o **hartă a porturilor** care poate sau nu să facă parte din harta memoriei.

Singura deosebire esențială față de locațiile de memorie este *legătura fizică pe care porturile o asigură cu exteriorul*; pentru microprocesor, de multe ori, această legătură fizică este transparentă și nesemnificativă. Organizarea porturilor comparativ cu organizarea memoriei constituie un bun exemplu de ilustrare a relației dintre *logic* și *fizic* în organizarea sistemelor de prelucrarea informației.

În sfârșit, componenta hardware a calculatorului comportă un set de legături specifice; acestea se realizează printr-o așa numită **magistrală**: un set de conexiuni fizice între blocuri prin care informația care circulă are o semnificație prestabilită. Sistemele la care ne referim au o magistrală unică care

le caracterizează; din punct de vedere funcțional, există trei componente ale acestei magistrale, marcate cu culori distincte și în Figura 1.1:

1. *Magistrala de date*, bidirecțională, permite circulația datelor (operanzi/rezultate), a instrucțiunilor și chiar a adreselor.

2. *Magistrala de adrese*, unidirecțională, permite microprocesorului să localizeze informația în MEMORIE sau în DISPOZITIVELE DE INTRARE/IEȘIRE; pe această magistrală circulă numai adrese.

3. *Magistrala de control* permite circulația, bidirecțională, a semnalelor de comandă și control de la/la microprocesor, în calitatea sa de UNITATE CENTRALĂ.

B. Din punct de vedere ***software***, a doua componentă definitorie a calculatorului, definirea rezultă practic din considerentele anterioare: o serie de programe organizate în moduri specifice.

Se poate face o clasificare în două categorii de software:

1. *Sistemul de operare*: totalitatea programelor care permit utilizatorului accesul complet la resursele sistemului. Programele, cu denumiri specifice după funcțiile realizate, asigură, în primul rând, accesul la echipamentele periferice: tastatură, afișaj, memorie de masă etc. Este evident, așadar, că fără un sistem de operare, chiar minim, calculatorul este o cutie inutilă. Sistemul de operare poate fi rezident (permanent în memoria internă) sau încărcabil dintr-o memorie externă (operație denumită „bootstrap").

2. *Software de utilizator* alcătuit din totalitatea programelor folosite pentru îndeplinirea unor sarcini specifice.

Prezentarea acestor noțiuni și definirea lor ne permit câteva concluzii care să facă o delimitare asupra conceptului de microprocesor așa cum este el înțeles în volumul de față:

- Microprocesorul constituie Unitatea centrală de prelucrare a unui sistem având schema bloc funcțională din Figura 1.1. El concentrează și funcția de prelucrare și pe cea de comandă.
- Toate celelalte componente ale sistemului nu au putere de decizie. Memoria nu controlează și nici nu e necesar să controleze semnificația informației pe care o deține și modul în care este organizată.
- Legătura dintre blocuri este asigurată de o magistrală unică cu trei componente funcționale; pe magistrala de date circulă toate tipurile de informații.

- Funcționarea sistemului se face pe baza unor programe alcătuite din secvențe de instrucțiuni. **Acestea sunt citite din memorie de către microprocesor, recunoscute și apoi executate** [12].

În aceste condiții, vom introduce și noțiunea de **arhitectură**: **totalitatea atributelor sistemului (în cazul de față, microprocesorul) care sunt disponibile („vizibile") utilizatorului (ca de pildă: registrele, modurile de adresare, tipurile de transferuri de date, modul de organizare logică a memoriei, tehnicile de intrare/ieșire, setul de instrucțiuni etc)**.

Se observă că, din punctul nostru de vedere, noțiunea de **arhitectură** nu se confundă cu cea de **structură**. Putem acum să precizăm și o delimitare a problemelor tratate în volumul de față: vom dezvolta aproape exclusiv concepte legate de arhitectură și vom aminti extrem de sumar, numai unde este absolut necesar, aspectele legate de structură. Pe de altă parte, nu ne limităm la setul de instrucțiuni (deci nu vom dezvolta tehnicile de programare în limbaje de asamblare) pentru că acesta constituie doar o latură a arhitecturii.

Am introdus noțiunea de *arhitectură* și precizarea în legătură cu relația *arhitectură – structură* încă din 1994 în lucrarea [4]. Definiții și precizări similare apar mai târziu, în 2010, în lucrarea [13].

1.2 DEFINIȚII. MICROPROCESOARE CISC și RISC

Definiția microprocesorului ca UNITATE CENTRALĂ DE PRELUCRARE presupune implicit că sistemul din care face parte este un (micro/mini) calculator, deci *un sistem de calcul*. Putem extinde însă noțiunea și asupra **sistemelor de comandă și control** (de tip **controler**), ceea ce mărește aria de cuprindere a noțiunii de microprocesor.

O delimitare necesară stabilirii cadrului dezvoltărilor din capitolele următoare este legată de câteva posibile clasificări ale noțiunii de microprocesor. Se pot face clasificări din mai multe puncte de vedere:

a) După lățimea magistralei de date: microprocesoare pe 8, 16, 32 sau pe 64 biți.

b) După tipul de sarcini eficient realizabile:

• microprocesoare de uz general **(µPUG**), nespecializate;

• microprocesoare specializate, ca de pildă:

- **procesoare grafice** (*Graphics Processing Unit* – **GPU**) destinate inițial pentru procesarea digitală a imaginilor și accelerarea graficii pe calculator;
- **procesoare tensoriale** (*Tensor Processing Unit* – **TPU**), dezvoltate inițial de Google pentru accelerarea procesării în învățarea automată (*machine learning*) permițând operații cu tensori de mari dimensiuni;
- **procesoare digitale de semnal** (*Digital Processing Processor* – **DSP**), specializate pentru algoritmii specifici prelucrării semnalelor (FFT, produse de corelație, filtre digitale, calcul matriceal etc.); exemplu: Texas Instruments TMS 320.

c) După principiile de bază ale arhitecturii care guvernează funcționarea:

- **procesoare cu set complex de instrucțiuni (CISC)** numite microprocesoare standard (mult timp erau dispozitivele numite generic „microprocesoare”);
- **procesoare cu set redus de instrucțiuni (RISC)** [3], [8].

Pentru orice tip de microprocesor vom folosi și denumirea prescurtată de **procesor**.

Precizările pe care le-am făcut în acest capitol ne permit să delimităm deja destul de clar scopul volumului de față: **studiul conceptelor de bază ale arhitecturii (micro)procesoarelor standard, de uz general CISC și RISC**.

1.3 ARHITECTURI ALTERNATIVE: HARVARD

Arhitectura Von Neumann nu reprezintă singura modalitate de organizare a blocurilor funcționale ale unui sistem de calcul. Are avantajul de a fi foarte simplă și intuitivă, dar are și dezavantaje majore.

Principalul dezavantaj este conflictul între operanzi/rezultate și coduri de instrucțiuni pe magistrala de date unică. Așa cum vom detalia în Capitolul 10, desfășurarea în timp secvențială a instrucțiunilor implică timpi de așteptare suplimentari pentru a permite codurilor de instrucțiuni și operanzilor/rezultatelor aferente să nu genereze conflicte de transfer pe magistrală.

O soluție posibilă este oferită de **arhitectura Harvard** (Figura 1.2) – una dintre alternativele la arhitectura clasică Von Neumann [3].

Organizarea blocurilor funcționale implică separarea resurselor de memorie fizic, nu numai logic, în două părți: memorie de program și memorie de date. Cele două memorii sunt adresate separat și au propriile magistrale date astfel încât nu mai există riscul de conflicte pe magistrală și acțiunile de acces în memoria de program și în memoria de date pot avea loc (cvasi)simultan.

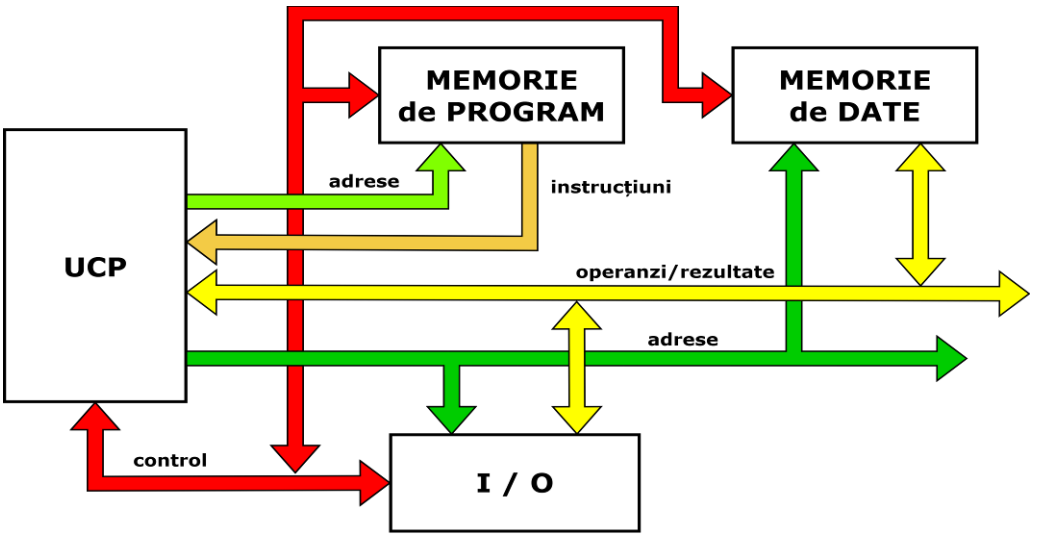

Fig. 1.2 Arhitectura Harvard

Astfel de organizări ale sistemelor de calcul sunt folosite în special de microcontrolere și procesoare specializate (de pildă procesoare digitale de semnal). În Capitolul 15 vom prezenta funcționarea unui dispozitiv concret, microcontrolerul din familia Intel 8051. Acest microcontroler are o memorie de

program, o memorie de date externă și o memorie de date internă, pe cipul gazdă.

Este interesant de observat că pentru o astfel de organizare a memoriei adresele nu mai sunt unice. O aceeași valoare de adresă poate identifica, în funcție de ținta validată, o locație în memoria de program, o locație în memoria de date (internă și/sau externă în cazul Intel 8051 – vezi Capitolul 15).

1.4 REPREZENTAREA INFORMAȚIEI ÎN SISTEMELE DIGITALE

Modul în care se prezintă informațiile în sistemele digitale în general (și în cele cu microprocesoare, în special) prezintă două aspecte distincte:

- Reprezentarea în interiorul sistemului, reunind convențiile de scriere a numerelor binare, de interpretare a codurilor etc (*reprezentarea internă*).
- Reprezentarea pentru utilizator, adică modul în care se pot face „vizibile" și interpreta informațiile din sistem (*reprezentarea externă*) [2], [9], [11].

Reprezentarea internă

Relativ la reprezentarea binară, singura posibilă în sistemele actuale cu microprocesoare, vom utiliza următoarele noțiuni generale:

- ***bit*** (prescurtat **b**) pentru o cifră binară **0** sau **1**;
- ***jumătate de octet*** sau ***nibble*** (prescurtat **n**) pentru o înșiruire de 4 biți;
- ***octet*** sau ***byte*** (prescurtat **B**) pentru o înșiruire de 8 biți;
- ***cuvânt*** sau ***word*** (prescurtat **w**) pentru o înșiruire de 2 octeți;
- ***cuvânt dublu*** sau ***double word*** (prescurtat **dw**) pentru o înșiruire de 4 octeți;
- prefixele:
  - **K** pentru $\mathbf{2^{10} \approx 10^3}$;
  - **M** pentru $\mathbf{2^{20} \approx 10^6}$;
  - **G** pentru $\mathbf{2^{30} \approx 10^9}$;
  - **T** pentru $\mathbf{2^{40} \approx 10^{12}}$.

În continuare, ne propunem o minimă trecere în revistă a convențiilor pentru reprezentarea internă a programelor, numerelor și datelor alfanumerice. În special în ceea ce privește numerele, ne vom limita la câteva tipuri de reprezentări pe care le considerăm strict necesare, pentru care vom defini doar regulile de interpretare a convențiilor.

a) Reprezentarea programelor

Așa cum arătam anterior, fiecare instrucțiune este reprezentată în memorie de un *cod binar*. **Formatul instrucțiunilor**, adică totalitatea cifrelor binare necesare pentru codificare, are, de regulă, drept cuantă de informație, octetul. Pentru fiecare instrucțiune există un număr prestabilit de octeți cu care e codificată (de pildă, pentru procesoarele de tip Intel x86 pe 32 biți, formatul instrucțiunilor are între 1 și 15 octeți).

b) Reprezentarea numerelor

1) *Reprezentarea întregilor fără semn în binar natural*

Este reprezentarea uzuală, „naturală" a numerelor binare.

■ **Exemplul 1.1**

număr binar cu 8 cifre:

$$\sum_{i=0}^{7} b_i \, 2^i \qquad \text{cu } b_i \in \{0, 1\}$$

Scrierea este pozițională, ca în orice bază de numerație, deci poziția fiecărui bit indică puterea lui 2 corespunzătoare. Numărul va fi scris forma $b_7b_6b_5b_4b_3b_2b_1b_0$

■

Vom denumi bitul cel mai puțin semnificativ (b_0 în cazul nostru) cu **lsb**, iar bitul cel mai semnificativ (în exemplul de mai sus, b_7) cu **msb** (folosind prescurtările de la exprimările în limba engleză).

2) *Reprezentarea întregilor cu semn în binar natural*

Semnul numărului este reprezentat de **msb** cu următoarea convenție:

msb = 0 semnifică număr **pozitiv**;
msb = 1 semnifică număr **negativ**.

Este ușor de observat că mărimea plajei de numere reprezentate în binar nu se schimbă, ci doar poziția față de zero. Astfel, pentru un număr fără semn cu 8 biți, plaja numerelor reprezentabile acoperă 256 de poziții, între **0** și **255**, în zecimal. Pentru un număr cu semn, plaja numerelor reprezentabile acoperă tot 256 de poziții, dar în intervalul este **-128** ÷ **+127**, presupunând **0** număr pozitiv.

Dacă pentru semn există o convenție unanim acceptată, în ceea ce privește mărimea numărului există mai multe convenții de reprezentare. Vom aminti cele mai răspândite trei convenții: *în mărime și semn*, *complement față de 1* și *complement față de 2*. Tabelul 1.1 prezintă reprezentarea numerelor **+5** și **-5** cu 8 cifre binare folosind cele trei convenții.

Regulile de reprezentare în aceste trei convenții pot fi rezumate astfel:

- Numerele pozitive se reprezintă identic în cele trei reprezentări.

- În „*mărime și semn*”, numerele negative diferă de cele pozitive numai prin bitul de semn.

- În „*complement față de 1*”, mărimea numărului negativ se obține din reprezentarea precedentă prin *complementare* bit cu bit; convenția pentru bitul de semn se păstrează.

- În „*complement față de 2*”, mărimea numărului negativ se obține din reprezentarea precedentă prin *adunarea* unei cifre binare **1** la **lsb**.

■ **Exemplul 1.2**

Tabelul 1.1

Tipul reprezentării	**+ 5**	**- 5**
„mărime și semn”	**00000101**	**10000101**
„complement față de 1”	**00000101**	**11111010**
„complement față de 2”	**00000101**	**11111011**

■

Se impun câteva observații:

• Reprezentarea „în complement față de 2” este preferată în sistemele digitale pentru facilitatea de a folosi un algoritm unitar la operațiile de adunare indiferent de semnul numerelor (sau altfel spus, adunarea și scăderea pot folosi aceeași implementare hardware).

• **Transportul** care apare între rangul unui număr binar și cel imediat superior în operațiile aritmetice (la scădere, îl vom numi „împrumut”), reprezintă o informație care este stocată în mod special în circuitele sistemelor digitale care efectuează astfel de operații. Microprocesoarele au locații de memorie internă numite **fanioane** special destinate stocării unor indicatoare ale desfășurării unei operații aritmetice. Fanionul de transport este, de regulă, prescurtat cu **C** (de la „*carry*”).

• O variantă deosebită a transportului este aceea care apare la rangul cel mai semnificativ al unui număr. El are o deosebită importanță deoarece poate semnala că numărul de biți afectat reprezentării rezultatului unei operații este insuficient: apare o **depășire**. Această informație este stocată de către microprocesor într-un fanion dedicat: **O** (de la „*overflow*"). După cum numărul are semn sau nu, se poate scrie că:

$$O = C_{msb\nearrow} \quad \text{sau} \quad C_{msb-1\,\nearrow\,msb},$$

în care am notat cu **msb-1** rangul imediat inferior celui mai semnificativ bit.

• În operațiile aritmetice apare deseori necesitatea de a reprezenta un număr dat **cu dublă precizie**. Aceasta înseamnă că numărul trebuie să rămână neschimbat ca valoare deși va fi reprezentat cu un număr dublu de cifre binare. Pentru numerele cu semn această așa numită „*extindere a numărului cu semn*" nu este banală, depinzând de convenția de reprezentare. Tabelul 1.2 prezintă exemplul extinderii unor numere cu semn reprezentate în complement față de 2, de la 8 la 16 biți.

■ **Exemplul 1.3**

Tabelul 1.2

	Reprezentare cu 8 biți	**Reprezentare cu 16 biți**
+ 1	00000001	0**0000000**00000001
- 1	11111111	1**1111111**11111111

■

Regulile de *extindere a numerelor cu semn, în complement față de 2* pot fi sintetizate astfel:
- Bitul de semn rămâne pe poziția cea mai semnificativă.
- Partea care reprezintă mărimea numărului va ocupa pozițiile cele mai puțin semnificative ale numărului extins.
- Restul pozițiilor din numărul extins se completează cu cifre binare identice cu cea care reprezintă semnul (**0** pentru numere pozitive și **1** pentru numere negative).

3) *Reprezentarea întregilor în „***zecimal codificat binar***" (***ZCB***)*: principiul folosit este de a reprezenta fiecare cifră zecimală separat, în binar natural, cu un număr de biți suficient pentru a acoperi plaja cifrelor zecimale. Cum această plajă cuprinde cifrele de la **0** la **9** este suficient un *nibble* pentru reprezentare.

Microprocesoarele folosesc două tipuri de reprezentări ZCB:

- Reprezentarea „**ZCB împachetat**” în care fiecare octet din memorie cuprinde câte două cifre zecimale, una pe *nibble*-ul mai puțin semnificativ și cealaltă pe *nibble*-ul superior. Plaja de numere zecimale acoperită de o reprezentare cu 8 biți se micșorează de la 256 la 100 de numere: **0 ÷ 99**.

- Reprezentarea „**ZCB neîmpachetat**” în care fiecare octet cuprinde o singură cifră zecimală pe *nibble*-ul mai puțin semnificativ. Restul cifrelor binare se completează cu **0**.

4) *Reprezentarea numerelor cu zecimale cu virgulă fixă*: se folosește principiul de a aloca un număr fix, prestabilit, de cifre binare pentru a reprezenta partea întreagă și respectiv partea zecimală a unui număr.

Se poate folosi fie reprezentarea în binar natural fie în ZCB. Pentru partea întreagă se folosește regula de reprezentare a numerelor întregi cu semn, iar pentru partea zecimală regula de reprezentare a întregilor fără semn. Dezavantajul major al acestei reprezentări constă în riscul ca partea întreagă să depășească numărul de biți alocat (caz în care reprezentarea devine incorectă) și/sau ca partea zecimală să nu „încapă” în spațiul prestabilit (caz în care se recurge la „*trunchierea*” sau „*rotunjirea*” numărului).

Modul de reprezentare folosește următoarele convenții:

- Se rezervă un șir de biți cu care se exprimă numărul total de cifre ale numărului care urmează să fie reprezentat.
- Se rezervă, apoi, un șir de biți în care se înscrie numărul de zecimale cu care se va reprezenta numărul.
- Urmează reprezentarea propriu-zisă a numărului înșiruind reprezentările pentru partea întreagă și cea zecimală fără o altă delimitare explicită între ele.

■ **Exemplul 1.4**

Un număr reprezentat în ZCB împachetat

În acest caz, cuanta de informație este *nibble*-ul. Să presupunem atunci că se alocă două *nibble*-uri pentru numărul total de cifre, un *nibble* pentru numărul de zecimale și, pentru omogenitate, un *nibble* pentru semn. Atunci, dacă în memorie există următoarea înșiruire de cifre binare:

0000 0100 0010 0000 1001 0110 0001 0101 ....

numărul reprezentat este **+ 96.15**

■

5) *Reprezentarea numerelor cu zecimale cu virgulă mobilă* (reprezentare **normalizată)**.

Două entități informaționale: *mantisa* **M** și *exponentul* **EXP**:

$$\textbf{număr binar} = M * 2^{EXP}$$
$$2^{-1} \leq M < 2^{0} .$$

Avantajul constă în unificarea reprezentării numerelor într-un format standard și, mai ales, în extinderea plajei numerelor reprezentate astfel încât riscul de depășire scade considerabil.

■ **Exemplul 1.5**

$$b_{31}.....b_{24}b_{23}.....b_0,$$

în care: - $b_{31} \div b_{24}$ reprezintă exponentul, având semnul în poziția b_{31}.
- $b_{23} \div b_0$ reprezintă mantisa cu semnul la b_{23}.

Plaja numerelor reprezentabile în acest fel este $M * 2^{\pm 128}$, față de plaja numerelor reprezentabile *cu virgulă fixă* utilizând aceeași patru octeți (presupunând că ar fi întregi cu semn) care este $\pm 2^{31}$).

■

c) Reprezentarea datelor alfanumerice

Vom înțelege prin *date alfanumerice* sau *caractere* oricare dintre semnele care pot fi tipărite de la tastatura unui calculator. Există, în afară de litere şi cifre, o multitudine de simboluri care pot fi folosite. Din acest motiv se vorbește despre *seturi de caractere*, adică grupuri minime de simboluri considerate suficiente pentru a asigura o editare cât mai completă a unui text. De asemenea, este evident că pentru fiecare caracter se va folosi o reprezentare binară, **un cod**, cu care caracterul (dintr-un set prestabilit) este în relație biunivocă.

Convențiile pentru codificarea caracterelor nu sunt unanim acceptate. Există totuși un standard folosit practic de sistemele de calcul numit **ASCII** (de la **A**merican **S**tandard **C**ode for **I**nformation **I**nterchange) cu care se codifică următorul set de caractere:

- 26 de litere mari ale alfabetului latin;
- 26 de litere minuscule corespunzătoare;
- 10 simboluri numerice: **0** ÷ **9**;
- 20 de simboluri speciale adiționale: **+**, **-**, **(**, **)**, **[**, **]**, **{**, **}**, ***** , **#** , **$** etc.

Cum numărul total este mai mic de 128, rezultă că se poate realiza o codificare cu 7 biți; se va folosi deci câte un octet la care **msb** joacă un rol special, acela de **bit de paritate**. Convenția folosită este următoarea:

msb = 0 dacă codul are un număr *par* de cifre binare **1**;

msb = 1 dacă codul are un număr *impar* de cifre binare **1**;

■ **Exemplul 1.6**

Primele litere ale alfabetului, majuscule, au următoarele coduri:

A: **0**1000001;
B: **0**1000010;
C: **1**1000011 etc.

■

Microprocesoarele stochează, de regulă, bitul de paritate într-un fanion dedicat (**P**) folosit, de pildă, pentru o verificare simplă a corectitudinii transmisiei datelor pe magistrale sau în conversația cu perifericele.

Reprezentarea externă

Reprezentarea externă se referă la modul în care informația prelucrată de un calculator apare utilizatorului (programatorului). Desigur, se pune problema a trei tipuri de informații, ca și la reprezentarea internă.

a) Pentru **codurile instrucțiunilor** se vor folosi *mnemonice* care sunt abrevierile sugestive pe care, de regulă, fabricantul le impune și pe care limbajul de asamblare le folosește.

b) Pentru **numere** se utilizează mai multe tipuri de reprezentări:

- Reprezentarea *binară* care este imagine fidelă a conținutului locațiilor de stocare a informațiilor; interpretarea este greoaie dat fiind numărul mare de cifre binare implicate. O adaptare la interpretarea familiară utilizatorului este reprezentarea ZCB.
- Reprezentarea *octală* care transformă numerele binare în baza de numerație 8.
- Reprezentarea *hexazecimală*: un simbol reprezentând o cifră în baza de numerație 16 înlocuiește 4 cifre binare. Caracterele folosite sunt cifrele **0** ÷ **9** și literele **A** ÷ **F**. Convenim să utilizăm litera **H** ca sufix pentru numerele reprezentate în hexazecimal (de pildă **1199H**) sau prefixul **0x** (de pildă **0x1199**).

c) Pentru **caractere** se vor folosi chiar simbolurile cu care ele sunt individualizate. Programele utilitare folosite pentru examinarea conținutului locațiilor de stocare a informațiilor fac conversia **ASCII → simbol caracter** dacă programatorul stabilește că semnificația informației vizate impune această conversie.

Se poate conchide că reprezentarea externă depinde în mod esențial de semnificația pe care utilizatorul o conferă conținutului locațiilor de stocare a informației. Utilitarele folosite vor interpreta șirurile de cifre binare ca numere, coduri corespunzând unor mnemonice sau coduri corespunzând unor date alfanumerice după cum decide programatorul.

1.5 CONVENȚII PENTRU NOTAȚII

Vom trece în revistă, în continuare, cele mai importante convenții pentru notațiile care vor fi folosite în următoarele capitole. Anumite notații specifice vor fi explicate în momentul introducerii lor. Aceste notații vor fi folosite pentru **descrierea formală a semanticii acțiunilor microprocesoarelor** [5], [6], [14].

În general, vom folosi reguli de descrierea sintaxei folosind *forma Backus-Naur* (BNF). Astfel, notațiile utilizate se împart în mai multe categorii:

1. Neterminali

r	un registru oarecare;
r8	un registru de 8 biți;
r16	un registru de 16 biți;
r_i , r_j	registre individualizate, diferite;
mem	o locație de memorie oarecare (sau mai multe locații succesive);
mem8	o locație de memorie de un octet;
mem16	o locație de memorie de 16 biți (pot fi două locații succesive dacă formatul este octetul);
mem32	o locație de memorie de 32 de biți (pot fi patru locații succesive dacă formatul este octetul);
mem_i	o locație de memorie individualizată (în scopul de a o deosebi de alte locații de memorie);
adr	o adresă oarecare;
adr8	o adresă pe 8 biți;
adr16	o adresă pe 16 biți;
adr24	o adresă pe 24 de biți;
adr_i	o adresă individualizată (în scopul de a o deosebi de alte adrese);
(r)	conținutul unui registru oarecare;
(r_i, r_j)	conținutul a două registre concatenate;
$(r)_l$	conținutul jumătății inferioare (mai puțin semnificativă) a unui registru;
$(r)_h$	conținutul jumătății superioare (mai semnificativă) a unui registru;

((r))	conținutul unei locații de memorie a cărei adresă se află într-un registru (**adresare indirectă**);
(mem)	conținutul unei locații de memorie oarecare;
adr$_l$	jumătatea inferioară a unei adrese;
adr$_h$	jumătatea superioară a unei adrese;
data	un operand oarecare;
data8	un operand de 8 biți;
data16	un operand de 16 biți;
disp8	un deplasament pe 8 biți;
disp16	un deplasament pe 16 biți;
port	un port de intrare/ieșire oarecare.

Observații:

- Parantezele rotunde se folosesc pentru a desemna **conținutul** unui registru, a unei perechi de registre, a unei locații de memorie sau a unui port. Sunt necesare pentru face deosebirea categorială între entitatea fizică (registru, locație de memorie, port) și conținutul acesteia.
- Parantezele rotunde duble semnifică **adresare indirectă**.
- Indicii atașați unei notații individualizează elementul respectiv, fără a-l denumi efectiv, cu scopul de a-l diferenția de alte elemente de același tip.
- Indicii **l** și **h** sunt rezervați desemnării jumătăților entităților respective.

2. Terminali:

Se referă la **numele** date unor elemente care capătă astfel o identitate bine precizată (de regulă sunt numele folosite de fabricantul procesorului respectiv):

R1, R2, A, AX, SP, A6, Dn, An	nume de registre;
(AX)	conținutul registrului **AX**;
(R1, R2)	conținutul perechii de registre **R1** și **R2**;
((SP))	conținutul locației de memorie a cărei adresă se află în registrul **SP**;
MEM, **MEM1**	nume de locații de memorie;
ADR, **ADRn**	nume de adrese.

Observații:

- Regulile folosirii parantezelor rămân valabile ca în cazul neterminalilor.
- Nu se folosesc indici. Cifrele care apar în cazul unui registru sau al unei locații de memorie fac parte din numele entității respective.

3. Operatori:

←	atribuire;
↕	concatenare;
not	complementare (negație);
∀	operația logică SAU;
&	operația logică ȘI;
⊕	operația logică SAU EXCLUSIV;
+	adunare;
-	scădere;
*	înmulțire;
DIV	împărțire între numere întregi;
MOD	restul împărțirii între numere întregi.

4. Alte simboluri:

[ ]	încadrează elemente de sintaxă opționale;
|	delimitează elemente de sintaxă alternative.

Nu trebuie confundate convențiile pentru descrierea formală a semanticii acțiunilor microprocesoarelor, descrise mai sus, cu regulile de sintaxă folosite de un anumit limbaj de asamblare.