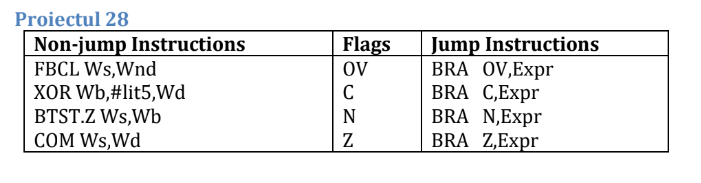
**Documentatie proiect**

**Calculatoare Numerice**

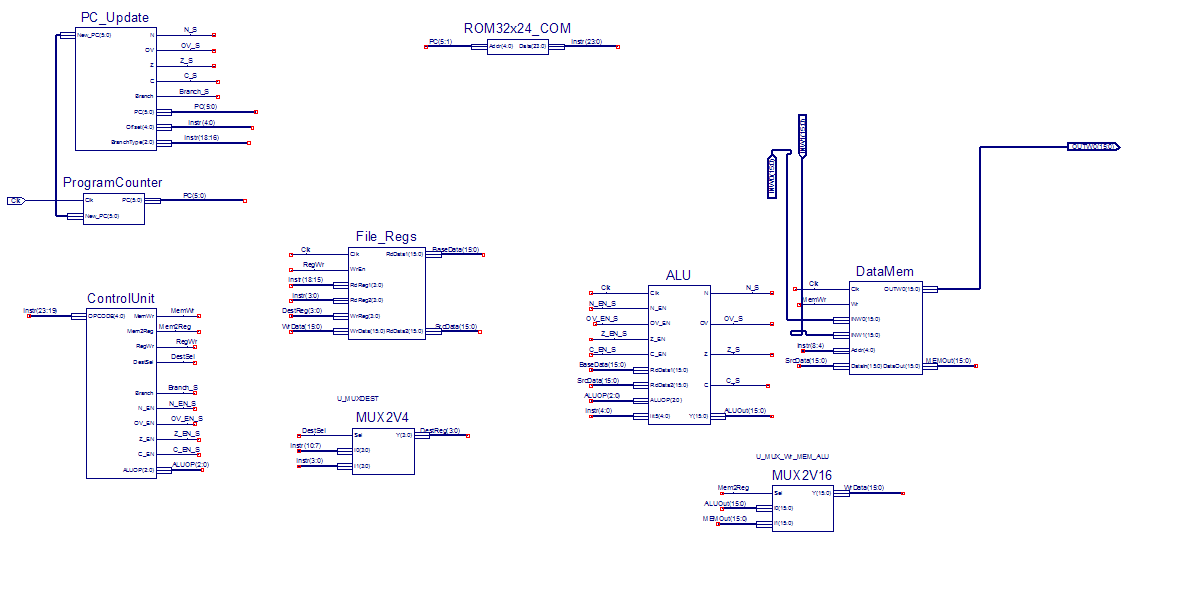
**PIC24**

**Matei Dragos Catalin**

**CR 3.2A**



**Schema Bloc**

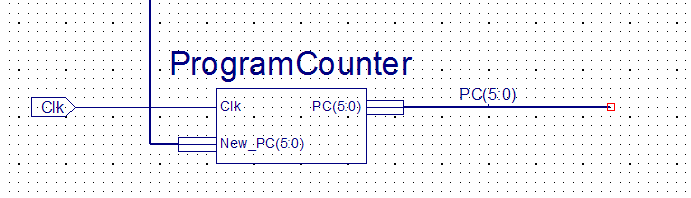


**Blocurile din cadrul procesorului**

ProgramCounter

 **Acest bloc este responsabil pentru controlul bistabilului, care are rolul de a actualiza valoarea curentă a numărătorului de program în momentul în care se primește impulsul de tact. Bistabilul reține și stabilește noua valoare a numărătorului de program, asigurând astfel actualizarea corectă a acestuia la fiecare ciclu de ceas.**

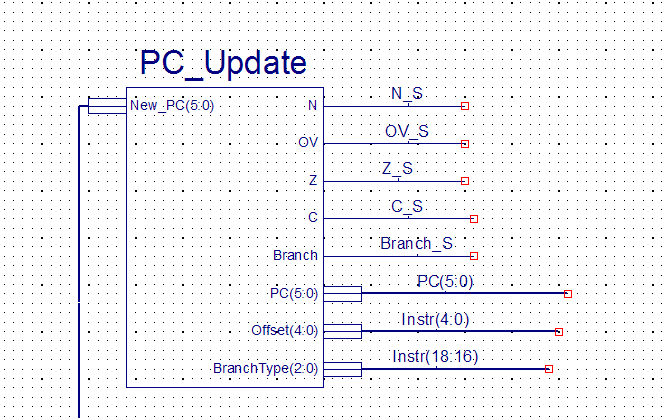
 **Valoarea care este aplicată la bistabil este furnizată de către blocul PC\_Update. Acest bloc calculează sau actualizează valoarea numărătorului și o trimite bistabilului pentru a fi stocată și utilizată în ciclurile de execuție ulterioare. Prin intermediul acestei interacțiuni, bistabilul se asigură că numărătorul de program reflectă corect modificările necesare conform datelor primite.**



PC\_Update

 **Acest bloc este responsabil pentru actualizarea valorii numărătorului de program. Modificarea poate avea loc fie prin avansarea la următoarea instrucțiune (de exemplu, PC=PC+2), fie prin executarea unei instrucțiuni de salt.**

 **Blocul primește semnalele de flag (N, OV, Z, C) care sunt utilizate pentru a decide dacă trebuie să fie aplicate instrucțiuni de salt condiționat. În plus, este inclus semnalul de Branch, care controlează dacă saltul este condiționat sau necondiționat. Blocul gestionează și un semnal de deplasament pentru calculul adresei de salt, precum și un semnal pe 3 biți numit BranchType, care permite diferențierea între cele cinci tipuri de instrucțiuni de salt.**

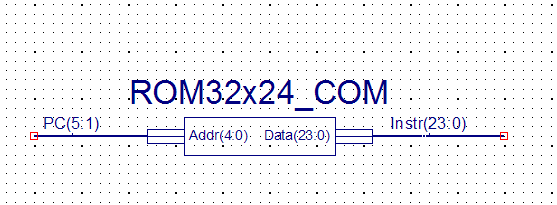


**ROM32x24\_X**

 **Acest bloc este echipat cu o memorie ROM (Read-Only Memory) care este utilizată pentru a stoca instrucțiunile programului. Rolul principal al acestei memorii este de a păstra permanent codul programului, asigurând accesul rapid la instrucțiuni necesare pentru execuția programului.**

 **Memoria ROM este adresată folosind o adresă de 5 biți, care este generată de numărătorul de program. Această adresă permite accesul la un spațiu de memorie care poate conține până la 32 de instrucțiuni diferite. Fiecare instrucțiune este stocată pe 24 de biți, asigurând o capacitate adecvată pentru codul programului.**

 **Instrucțiunile stocate în memorie sunt apoi utilizate în diverse faze ale procesului de execuție al programului, inclusiv decodificarea și execuția efectivă a instrucțiunilor. Astfel, acest bloc de memorie ROM joacă un rol esențial în facilitarea funcționării corecte și eficiente a procesorului, asigurând că instrucțiunile programului sunt disponibile pentru utilizare în timpul execuției.**



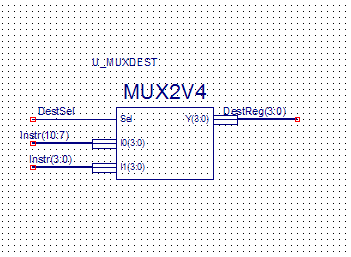
**Multiplexoarele de selectare regiștrii bază și destinație**

 **Modulul MUX\_D\_dddd este responsabil pentru selectarea segmentelor specifice de biți dintr-o instrucțiune, așa cum sunt descrise de conținutul memoriei ROM. Acest modul îndeplineste urmatoare functie:**

* **MUX\_D\_dddd**: Acest multiplexor se ocupă cu selectarea biților care indică registrul destinație. Instrucțiunea în ROM conține un câmp care specifică care registru va fi utilizat ca destinație pentru rezultatul operațiunii. MUX\_D\_dddd extrage acești biți și îi trimite către blocul de regiștri pentru a identifica și utiliza registrul corect în timpul execuției instrucțiunii.

  **MUX\_D\_dddd este controlat de unitatea de control a procesorului. Unității de control îi sunt furnizate semnalele DestSel și BaseSel, care determină modul în care acești multiplexori selectează segmentele de biți din instrucțiune. Semnalul DestSel controlează selecția pentru registrul destinație, iar semnalul BaseSel controlează selecția pentru registrul sursă, conform specificațiilor din tabelul de semnale al blocului de control.**

 **Astfel, MUX\_D\_dddd joacă un rol crucial în identificarea și transmiterea corectă a registrelor implicate în operațiunile ALU, contribuind la corectitudinea și eficiența execuției instrucțiunilor în procesor.**



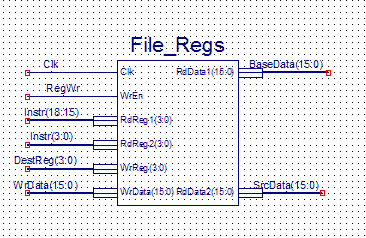
**File\_Regs**

 **Acest bloc este responsabil pentru gestionarea și stocarea registrelor procesorului. Conform designului, există un total de 16 registre, fiecare având o lățime de 16 biți. Aceste registre sunt esențiale pentru operarea procesorului, deoarece ele stochează datele și valorile temporare necesare pentru executarea instrucțiunilor.**

 **Blocul îndeplinește următoarele funcții:**

* **Furnizarea operanzilor necesari pentru execuția instrucțiunilor. În acest scop, blocul returnează datele stocate în registre prin intermediul semnalelor RdData1(b) și RdData2(s). Aceste semnale oferă valorile registrului de bază și al registrului sursă, pe care le utilizează ALU pentru efectuarea operațiunilor necesare.**
* **Selectarea registrelor pentru citire și scriere. Utilizând semnalele de selecție RdReg1(b) și RdReg2(s), blocul selectează registrul care va fi citit pentru operanzii instrucțiunii. Aceste selecții permit blocului să acceseze datele corecte din registrele procesorului.**
* **Scrierea de date într-un registru specificat. Blocul poate scrie o valoare într-un registru selectat, folosind semnalul WrReg pentru a selecta registrul țintă. Valoarea care trebuie scrisă este transmisă prin semnalul WrData. Operațiunea de scriere este activată de semnalul de scriere RegWr, care se activează la impulsul de tact, asigurând astfel actualizarea corespunzătoare a registrului.**

 **În rezumat, acest bloc de registre este crucial pentru funcționarea procesorului, asigurând atât accesul la datele necesare pentru execuția instrucțiunilor, cât și actualizarea corespunzătoare a registrelor în timpul procesului de scriere.**



**ALU**

 **Acest bloc este responsabil pentru aplicarea unor operații aritmetico-logice esențiale asupra datelor. Instrucțiunile pe care acest bloc le poate executa includ:**

* **FBCL Ws, Wnd:**
  + **Instrucțiunea FBCL (Fast Branch Conditional Logic) este utilizată pentru a efectua salturi condiționate rapide pe baza valorii unui registru și a semnalelor de flag. În această instrucțiune, Ws reprezintă registrul care conține valoarea testată, iar Wnd este registrul în care se verifică dacă saltul este condiționat corect.**
* **XOR Wb, #lit5, Wd:**
  + **Instrucțiunea XOR efectuează o operație de XOR (exclusive OR) între un registru Wb și o valoare literară de 5 biți, rezultatul fiind stocat în registrul Wd. Această operație logică este utilizată pentru a modifica sau verifica anumite biți specifici în registru.**
* **BTST.Z Ws, Wb:**
  + **Instrucțiunea BTST (Bit Test) verifică starea unui bit specific din registrul Ws. Operația este realizată folosind valoarea din Wb pentru a specifica bitul de testat. Semnalul Z (Zero) este utilizat pentru a indica dacă bitul verificat este zero.**
* **COM Ws, Wd:**
  + **Instrucțiunea COM (Complement) calculează complementul bitar al valorii din registrul Ws și stochează rezultatul în registrul Wd. Această operație este utilizată pentru a inversa toți biții valorii din Ws.**

 **Selecția operației specifice care va fi executată de acest bloc este determinată de un semnal extern, denumit ALUOP, care are o lățime de 3 biți. Semnalul ALUOP este generat de unitatea de control și indică tipul exact de operație aritmetico-logica pe care blocul trebuie să o efectueze.**

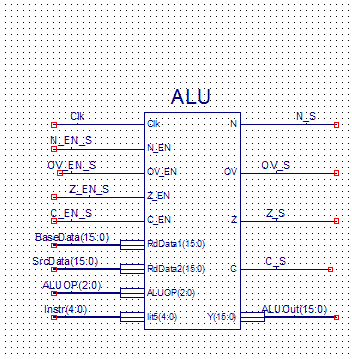
 **Blocul primește doi operanzi pe 16 biți, care sunt utilizați pentru efectuarea operației specificate. După ce operația este completă, blocul returnează rezultatul calculului.**

 **În plus, blocul actualizează patru biți de stare care reflectă rezultatul operației efectuate:**

* **N (Negative): Indică dacă rezultatul operației este negativ.**
* **OV (Overflow): Indică dacă a avut loc o depășire de capacitate în timpul operației aritmetice.**
* **Z (Zero): Semnalează dacă rezultatul operației este zero.**
* **C (Carry): Indică dacă a avut loc un carry (transport) în timpul operației aritmetice.**

 **Actualizarea acestor biți de stare se face conform semnalelor de permisiune date de unitatea de control, care sunt N\_EN, OV\_EN, Z\_EN, și C\_EN. Aceste semnale controlează dacă bitii de stare respectivi trebuie să fie actualizați pe baza rezultatului operației efectuate.**

 **În esență, acest bloc efectuează o serie de operații aritmetice și logice, oferind atât rezultatul calculului, cât și actualizarea biților de stare esențiali pentru procesul decizional al procesorului.**

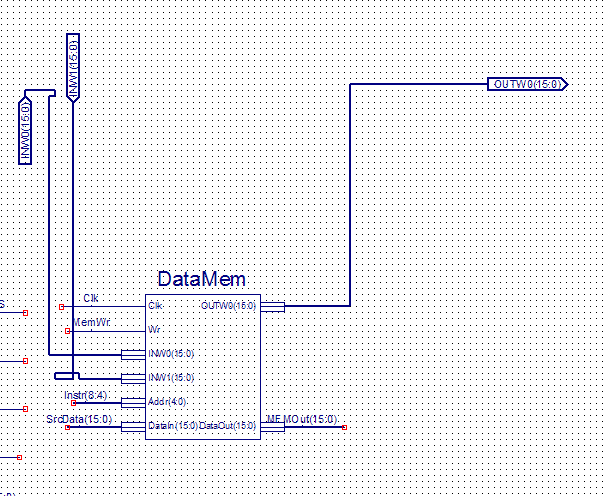


**DataMem**

**Memoria de date a procesorului este organizată într-o structură de 16 locații, fiecare având o lățime de 16 biți, plus 3 locații speciale destinate interacțiunii cu mediul extern. Iată o detaliere a componentelor și funcționalităților sale:**

1. **Structura Memoriei:**
   * **Memoria de date este compusă dintr-un aranjament de 16 locații, fiecare având o lățime de 16 biți. Aceasta permite stocarea și manipularea eficientă a datelor temporare necesare în timpul execuției programelor.**
   * **În plus față de cele 16 locații standard de memorie, există 3 locații speciale dedicate funcțiilor specifice de intrare și ieșire (I/O).**
2. **Locații Speciale:**
   * **Adresa 0x1020: INW0**
     + **Această locație specială este utilizată pentru a primi date de intrare din mediul extern. Semnalul INW0 permite procesorului să acceseze datele care sunt primite de la dispozitive externe sau alte surse de intrare.**
   * **Adresa 0x1022: INW1**
     + **Similar cu INW0, această locație specială este folosită pentru a citi date de intrare suplimentare. INW1 este destinată altor fluxuri de date externe, oferind astfel flexibilitate în interacțiunea procesorului cu lumea exterioară.**
   * **Adresa 0x1024: OUTW0**
     + **Această locație este utilizată pentru a trimite date către mediul extern. Semnalul OUTW0 permite procesorului să scrie date în dispozitive externe sau să le transmită altor module, facilitând comunicarea cu lumea exterioară.**
3. **Operațiuni de Scriere:**
   * **Scrierea în memoria de date se realizează prin aplicarea unui semnal de scriere, denumit MemWr, generat de unitatea de control a procesorului.**
   * **Pentru a efectua o scriere, trebuie specificată adresa de memorie la care se dorește scrierea, împreună cu un cuvânt de date de 16 biți care urmează să fie scris în acea locație. Semnalul MemWr activează operațiunea de scriere și asigură că datele sunt plasate corect în locația de memorie selectată.**
4. **Operațiuni de Citire:**
   * **Memoria de date este capabilă să returneze (citească) un cuvânt de date selectat prin semnalul de adresă.**
   * **Atunci când se efectuează o operație de citire, procesorul specifică adresa de memorie de la care dorește să recupereze datele. Memoria de date răspunde la această cerere, furnizând cuvântul de 16 biți stocat la adresa respectivă.**

**În esență, memoria de date a procesorului este o componentă crucială care gestionează atât datele interne, cât și interacțiunile externe prin locațiile speciale dedicate. Aceasta permite procesorului să efectueze operațiuni de citire și scriere esențiale pentru funcționarea eficientă a sistemului.**

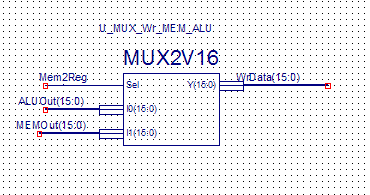


**MUX\_Mem2Reg**

**Acest bloc este responsabil pentru determinarea și selectarea cuvântului care va fi scris în registrul destinație după finalizarea execuției unei instrucțiuni. Blocul îndeplinește următoarele funcții esențiale:**

1. **Funcționalitatea Generală:**
   * **Scopul principal al acestui bloc este de a decide care cuvânt va fi transferat în registrul destinație, în funcție de rezultatul operațiunii curente. Acest cuvânt poate proveni fie din memoria de date, fie din rezultatul calculat de Unitatea Aritmetico-Logică (ALU).**
2. **Sursele de Date:**
   * **Memoria de Date:** Blocul poate selecta un cuvânt de 16 biți care a fost obținut printr-o operație de citire din memoria de date. Acest cuvânt poate fi rezultatul unei cereri anterioare pentru date de intrare sau un rezultat de interacțiune cu medii externe, stocat la o anumită adresă de memorie.
   * **Rezultatul ALU:** Alternativ, blocul poate selecta rezultatul generat de ALU, care este rezultat al operațiunilor aritmetico-logice executate, precum adunări, scăderi, operații logice (AND, OR, XOR) etc.
3. **Semnalul de Comandă:**
   * **Controlul selectării sursei de cuvânt se realizează printr-un semnal de comandă generat de unitatea de control, cunoscut sub numele de Mem2Reg.**
   * **Semnalul Mem2Reg indică dacă sursa de cuvânt pentru registrul destinație va fi rezultatul obținut din memoria de date sau rezultatul ALU. Acesta joacă un rol crucial în determinarea modului în care datele sunt transmise și actualizate în registrul destinație.**
   * **Când semnalul Mem2Reg este activat într-o anumită manieră (de obicei, pe baza valorii sale binare), blocul selectează sursa de cuvânt corespunzătoare și asigură că valoarea corectă este scrisă în registrul destinație.**
4. **Procesul de Transfer:**
   * **După ce a fost selectată sursa de cuvânt (memorie de date sau ALU), blocul trimite această valoare către registrul destinație în cadrul procesorului. Aceasta înseamnă că registrul destinație va fi actualizat cu valoarea selectată, completând astfel procesul de execuție al instrucțiunii.**

**În concluzie, acest bloc asigură corectitudinea și eficiența în procesul de scriere a rezultatelor în registrul destinație, fiind controlat de semnalul Mem2Reg pentru a selecta între rezultatul obținut din memorie și rezultatul ALU.**



**ControlUnit**

**Unitatea de Control (ControlUnit)**

**1. Funcționalitatea Generală:**

* **Unitatea de Control este un component esențial al procesorului, responsabilă pentru gestionarea și coordonarea tuturor celorlalte blocuri funcționale. Aceasta îndeplinește rolul de "creier" al procesorului, asigurând că toate operațiunile sunt executate în conformitate cu instrucțiunile primite și că secvența de execuție este corectă.**

**2. Decodificarea Instrucțiunii:**

* **Un aspect central al Unității de Control este decodificarea instrucțiunilor. Fiecare instrucțiune este reprezentată de un cod de operație (OPCODE), care este un semnal de lățime de 5 biți. Acest OPCODE este extras din magistrala de instrucțiune și reprezintă codul binar ce identifică tipul și specificațiile instrucțiunii ce trebuie executată.**
* **Unitatea de Control utilizează acest OPCODE pentru a determina ce operațiuni trebuie să fie efectuate și care semnale trebuie să fie activate. Aceasta decodificare este esențială pentru a traduce instrucțiunile din codul binar într-o serie de acțiuni și semnale care controlează funcționarea blocurilor procesorului.**

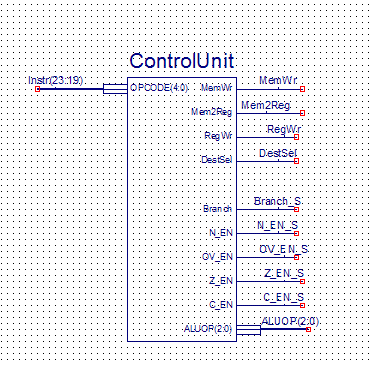
**3. Generarea Semnalelor de Control:**

* **Pe baza instrucțiunii decodificate, Unitatea de Control setează și gestionează semnalele de control necesare fiecărui bloc funcțional din procesor. Aceste semnale sunt esențiale pentru:**
  + **Configurarea și activarea corectă a unității aritmetico-logice (ALU), astfel încât să efectueze operațiile aritmetice și logice necesare.**
  + **Coordonarea transferului de date între memorie, registre și ALU.**
  + **Controlul fluxului de execuție, inclusiv gestionarea instrucțiunilor de salt și secvențialitatea execuției.**
* **Semnalele de control includ, dar nu se limitează la: semnale de scriere în memorie, semnale de citire din memorie, semnale de selectare a sursei de date, semnale de scriere în registre, semnale de activare a operațiunilor ALU și semnale pentru gestionarea ramificărilor (jumps) și salturilor condiționate.**

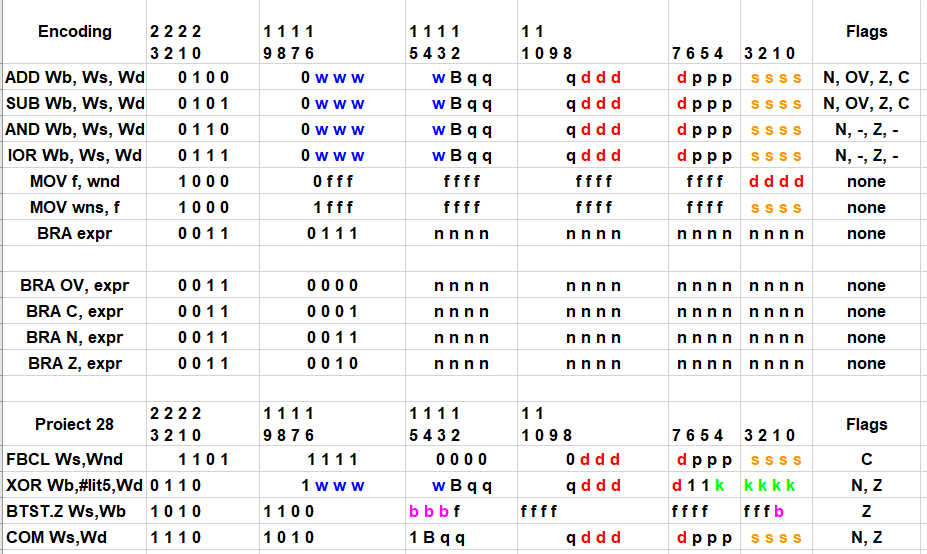
**4. Rolul în Secvențialitatea Execuției:**

* **Unitatea de Control garantează că instrucțiunile sunt executate într-o secvență corectă. Aceasta include:**
  + **Execuția secvențială a instrucțiunilor, asigurându-se că fiecare instrucțiune este completată înainte de a trece la următoarea.**
  + **Gestionarea instrucțiunilor de salt (branches), care pot modifica fluxul normal de execuție al programului. Unității de Control îi revine responsabilitatea de a actualiza numărătorul de program (Program Counter) conform instrucțiunilor de salt și de a se asigura că instrucțiunile sunt preluate și executate în ordinea corectă.**

**5. Importanța în Procesor:**

* **În concluzie, Unitatea de Control este "inima" procesorului, asigurând coordonarea eficientă și corectă a tuturor operațiunilor interne. Fără o Unită de Control eficientă, procesorul nu ar putea funcționa corect, deoarece nu ar putea interpreta și executa instrucțiunile în mod corespunzător.**
* 

**Tabelul de instructiuni**



**Tabelul de semnale de control**

