Simulator didactic al ciclului "Fetch-Decode-Execute"

Bucur Raul
Camil Laurențiu
Beciu Leonard

Mai 2023

1 Introducerea conceptului

Procesul *"fetch-decode-execute"* este un concept fundamental în arhitectura computerelor și descrie modul în care un procesor preia, interpretează și execută instrucțiunile programului.

Fetch (Preluare): În această etapă, procesorul preia următoarea instrucțiune din memoria principală (RAM) pe baza valorii conținute în registrul de instrucțiuni (*Instruction Pointer*). Instrucțiunea este apoi încărcată în registrul de instrucțiuni al procesorului.

Decode (Decodificare): În această etapă, procesorul interpretează instrucțiunea preluată și identifică operațiunile specifice pe care trebuie să le execute. Instrucțiunea este divizată în părți mai mici, cum ar fi codul operației și adresele sau datele asociate.

Execute (Execuție): În această etapă, procesorul execută operațiunile specificate de instrucțiunea decodificată. Aceasta poate implica operații aritmetice, logice, de stocare/încărcare în memorie, transferuri de date între registre etc. Rezultatele operațiunilor sunt stocate în registrele corespunzătoare.

După finalizarea etapei de executare, procesorul revine la etapa de preluare (fetch) și preia următoarea instrucțiune din memorie pentru a continua ciclul. Acest proces se repetă în mod continuu, permițând procesorului să parcurgă toate instrucțiunile programului într-o succesiune ordonată.

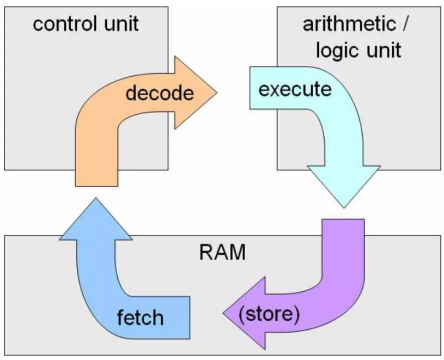

Figure 1: Vizualizarea operațiilor ciclului

2 "Fetch-Decode-Execute pentru procesoarele moderne

Procesoarele moderne sunt dispozitive electronice complexe, care reprezintă nucleul central al unui sistem de calcul. Ele sunt compuse din mai multe componente esențiale, care lucrează împreună pentru a efectua operații de calcul și gestionare a datelor într-un mod eficient.

Unitățile de procesare: Procesorul modern include unitatea centrală de procesare (CPU), care este responsabilă pentru executarea instrucțiunilor și efectuarea operațiilor de calcul. CPU-ul este alcătuit din mai multe unități specializate, cum ar fi unitatea de calcul aritmetic și logic (ALU), care efectuează operații matematice și logice asupra datelor, și unitatea de control, care coordonează operațiile întregului procesor.

Cache-uri: Cache-urile sunt memorii de acces rapid integrate în procesor, utilizate pentru a stoca date și instrucțiuni frecvent utilizate. Ele sunt organizate în ierarhii de niveluri, cum ar fi L1 (nivelul 1), L2 (nivelul 2) și L3 (nivelul 3), în funcție de distanța față de unitatea de procesare. Cache-urile reduc timpul de acces la datele și instrucțiunile frecvent utilizate, deoarece au o viteză de funcționare mai mare decât memoria principală.

Memorie principală: Procesorul interacționează cu memoria principală (RAM), care este o memorie mai lentă, dar cu o capacitate mult mai mare decât cache-urile. Memoria principală stochează date și instrucțiuni și este accesată prin intermediul unui sistem de adrese și interfețe specifice. Procesorul preia și stochează datele și instrucțiunile în memoria principală în timpul procesării.

Controlere și interfețe: Procesorul modern include controlere și interfețe pentru a gestiona comunicarea și interacțiunea cu alte componente ale sistemu-

lui. Acestea pot include controlere de stocare (cum ar fi pentru hard disk-uri sau SSD-uri), controlere de rețea, controlere grafice și alte dispozitive periferice. Controlerele și interfețele facilitează transferul de date și comunicarea între procesor și celelalte componente ale sistemului.

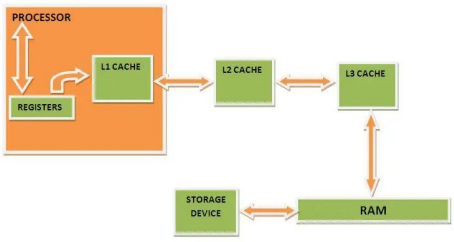

Figure 2: Modul de funcționare al memoriei cache

Arhitectura și setul de instrucțiuni: Procesoarele moderne sunt proiectate pe baza unei anumite arhitecturi, cum ar fi arhitectura x86 (folosită în procesoarele Intel și AMD) sau arhitectura ARM (folosită în procesoarele mobile). Fiecare arhitectură are propriul set de instrucțiuni, care reprezintă codurile și operațiunile pe care le poate executa procesorul. Arhitectura procesorului determină modul în care instrucțiunile sunt procesate, organizate și executate.

Un exemplu modern pentru acest procedeu de *fetch-decode-execute* poate fi dat prin intermediul procesoarelor Intel:

Procesoarele Intel moderne, cum ar fi cele din familia x86, utilizează o arhitectură complexă și sofisticată pentru a executa instrucțiuni. Iată o descriere mai detaliată a procesului *fetch-decode-execute* în cadrul unui procesor Intel modern:

Fetch (Preluare): În această etapă, procesorul preia instrucțiunile din memoria principală (RAM) pe baza valorii conținute în registrul de instrucțiuni. Procesoarele Intel folosesc o tehnică numită *preluare secvențială* (sequential fetch), în care instrucțiunile sunt preluate în ordine secvențială, adică începând de la o adresă de memorie și continuând cu următoarea adresă. În plus, procesoarele Intel utilizează o memorie cache specială numită *cache de instrucțiuni* (instruction cache) pentru a stoca instrucțiunile preluate recent, pentru a accelera accesul la memorie.

Decode (Decodificare): În această etapă, instrucțiunile preluate sunt decodate într-un format înțeles de procesor. Procesoarele Intel utilizează un circuit complex de decodificare care analizează codul instrucțiunii și identifică operațiunea specifică pe care trebuie să o execute, precum și registrele și datele implicate în această operațiune.

Execute (Execuție): În această etapă, procesorul execută operațiunile specifice identificate în etapa de decodificare. Procesoarele Intel folosesc o arhitectură RISC (*Reduced Instruction Set Computing*) internă, numită *arhitectura micro-ops* (micro-op architecture), în care instrucțiunile complexe sunt transformate într-un set de operații simple numite *micro-operații* (micro-ops). Aceste micro-operații sunt apoi executate în unitățile de execuție specializate, cum ar fi unități de calcul aritmetic și logic (ALU), unități de control în virgulă mobilă (FPU) sau unități de acces la memorie.

După finalizarea etapei de execuție, procesorul trece la următoarea instrucțiune și repetă ciclul *fetch-decode-execute* pentru aceasta.

3 Prezentarea și modul de utilizare ale aplicației

1. La stânga interfeței avem informații despre CPU:

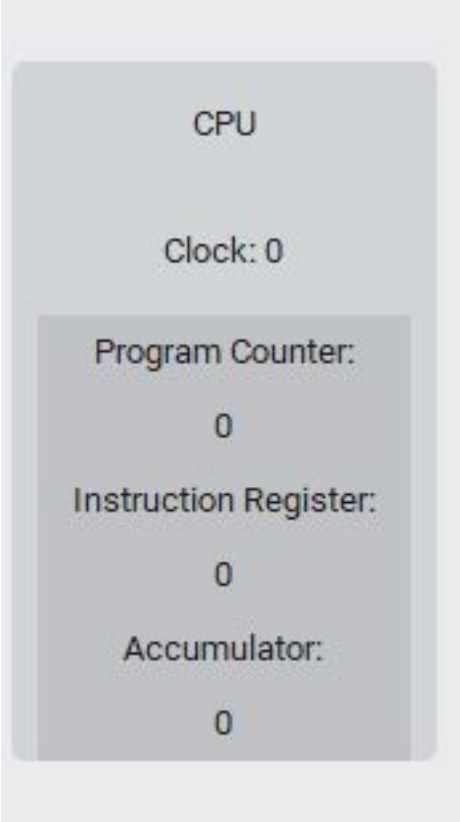

Figure 3: Căsuța CPU

Componente:

- Ceasul care se incrementează la fiecare apăsare a butonului *NEXT* (după ce pornim execuția cu *START*).
- Program Counter care indică adresa de la care urmează să citim la un anumit pas (se incrementează după *fetch*).
- Instruction Register unde se stochează instrucțiunea curentă.
- Accumulator unde se stochează valoarea obținută în urma executiei comenzilor.

2. În mijlocul interfeței avem o simulare simplificată a memoriei RAM sub forma unui tabel în care prima coloană reprezintă adresele iar a doua reprezintă valorile de la adresele respective:

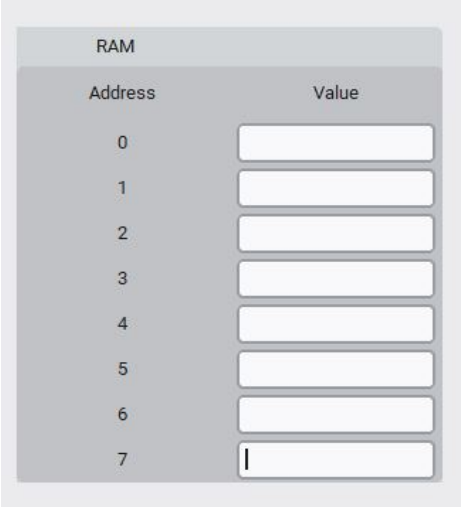

Figure 4: Căsuța RAM

În casuțe se introduc instrucțiuni sau valori numerice. După introducerea fiecăreia se apasă Enter pentru a se salva. Instrucțiunile suportate sunt: *load* {*number*}, *add* {*number*}, *sub* {*number*}, *mul* {*number*}, *div* {*number*}, *mod* {*number*}, *store* {*number*}, *jump* {*number*}, *for* {*no.of.instructions no.of.repeats*}, *if* {*number1 comparator number2 no.of.instructions*}.

{*number*} este o adresă de la 0 la 7 de la care se citește o anumită valoare. În cazul în care valoarea depășește intervalul, se va afișa o eroare.(la fel și {*no.of.instructions*})

{*no.of.repeats*} poate fi orice număr natural.

{*comparator*} poate fi unul dintre comparatorii acceptați de sintaxa instrucțiunii *if*. (Vezi *Modul de funcționare al comenzilor*)

Dacă este cazul să se ajungă la o instrucțiune cu doar o valoare numerică, se trece mai departe la următoarea instrucțiune până nu mai avem ce citi, moment în care se termină programul și se dezactivează butonul *NEXT*.

Modul de funcționare al comenzilor:

- *load* {*number*}: se citește valoarea de la adresa {*number*} și se stochează în acumulator.
- *add/sub/mul/div/mod* {*number*}: valoarea din acumulator se adună/scade/se înmulțește/se împarte/se ia restul împărțirii cu valoarea de la adresa {number}.
- *store* {*number*}: se stochează valoarea din acumulator la adresa {number}.

- *jump* {*number*}: se face *jump* cu program counter-ul la adresa {number} (se utilizează pentru a crea un ciclu).
- *for* {*no.of.instructions no.of.repeats*}: se simuleza un ciclu "for", instrucțiunile care se vor repeta find puse înaintea instrucțiunii *for*; modul de funcționare:
  - *no.of.instructions* reprezintă câte instrucțiuni dinaintea comenzii *for* vor intra în ciclu
  - *no.of.repeats* reprezintă numarul de repetări al ciclului *for*
- *if* {*number1 comparator number2 no.of.instructions*}: se compară valoarea de la adresa {*number1*} cu valoarea de la adresa {*number2*} folosindu-se {*comparator*}; dacă propoziția are valoare de adevăr se vor executa următoarele {*no.of.instructions*} comenzi, altfel se va sări peste ele; comparatorii acceptați sunt:
  - < - mai mic
  - <= - mai mic sau egal
  - > - mai mare
  - >= - mai mare sau egal
  - == - egal
  - ! = - diferit

3. În dreapta interfeței se află butoanele *START*, *RESET* și *NEXT* (care apare după ce se apasă *START*) și o casuță de text unde se afișează informații detaliate despre ciclul *fetch-decode-execute*: la care pas am ajuns, ce a făcut CPU la pasul respectiv, erori, end of program.

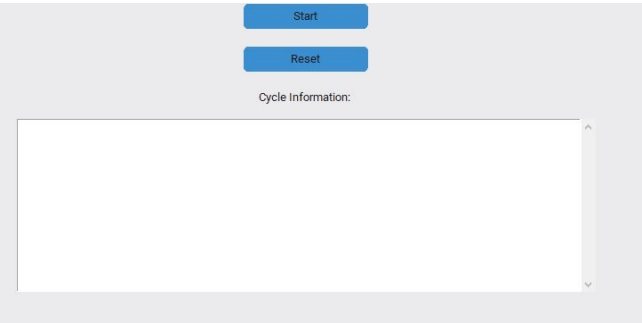

Figure 5: Butoanele START și RESET

4. Exemplu de utilizare:

- Se introduc instrucțiuni sau valori în casuțe, apăsând Enter după fiecare pentru a fi salvate.
- Se apasă *START* iar ciclul începe.

- Se apasă *NEXT* pentru a naviga prin cei trei pași care se repetă până la finalizarea executiei.
- Se apasă *RESET* când vrem să resetăm aplicația și să introducem noi valori.

(**Notă**: Toate elementele de interfață grafică folosite se regăsesc în pachetul "customtkinter". Pentru a putea rula programul această librărie trebuie instalată de către utilizator. Se recomandă folosirea IDE-ului "PyCharm". Fișierele cu extensia ".py" trebuie introduse într-un proiect în PyCharm, apoi se instalează pachetul "customtkinter" folosind una din următoarele metode:

- Metoda 1: Se poziționează cursorul pe cuvântul "customtkinter" de pe primia linie din fișierul main.py. Se apasă alt+enter și se selectează opțiunea "Install package customtkinter";
- Metoda 2: Se deschide un terminal PowerShell în proiect și se folosește comanda "pip install customtkinter==5.0".)

5. Exemple de secvențe de instrucțiuni (introduse în ordine):

- *load 6, add 7, store 6, jump 1, 0, 0, 1, 1*
- *load 5, 0, sub 6, store 1, jump 1, 1, 1*
- *load 3, mod 2, 2, 5*
- *load 7, if 7 > 6 1, sub 6, add 6, store 7, for 4 4, 7, 5*

Secvențele de instrucțiuni sunt la libera alegere, dar trebuie să respecte formatul, altfel se vor afișa erori și vom fi nevoiți să apăsăm *RESET*.

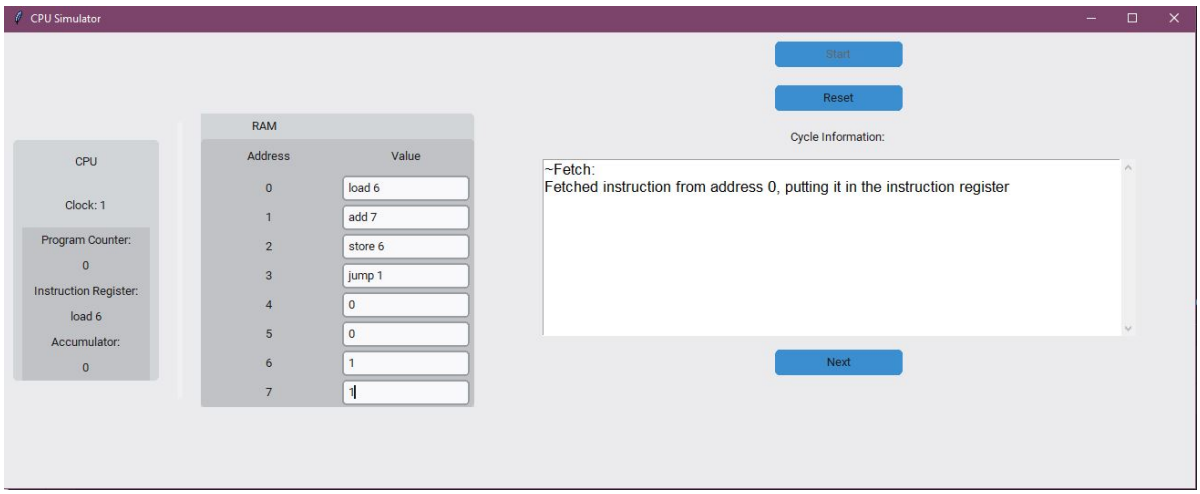

Figure 6: Exemplu de funcționare

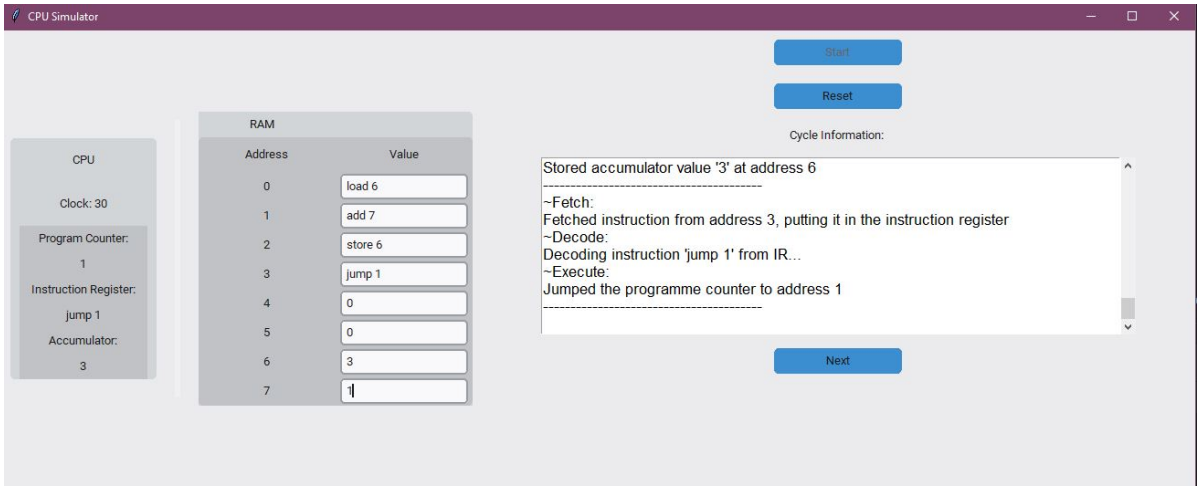

Figure 7: Exemplu de funcționare