**General Purpose Processor**

Fundamente de ingineria calculatoarelor

**Introducere**

Scopul acestui proiect este de a proiecta și implementa un procesor general purpose de 16 biți. Proiectul este împărțit în 3 faze: proiectare, simulare si proiectare software. Faza de proiectare constă în proiectarea setului de instrucțiuni personalizate și a arhitecturii CPU de nivel scăzut. Faza de simulare începe cu implementarea designului software folosind Verilog în ModelSim. Aceasta testează și depanează designul folosind tot ModelSim. Aceasta este urmată de faza de proiectare software, în care este scris un program pentru a demonstra funcționalitatea procesorului.

Din echipa au facut parte: Petcu Cosmin (Project Manager), Pitropu Andrei (Hardware Design), Pitic Sebastian (Hardware Design), Picior Alexandru (Software Design), Pascaru Tudor (Software Design) si Salavat Mihai (Tester QA + UX).

Contorizarea nivelului de lucru a fost facuta prin JIRA, fiecare parte principala (software, hardware si management) a fost un epic, ele avand task-uri corespondente topicului lor. Milestone-urile au fost: Design Hardware & Documentatie GP, Implementare General Purpose Processor, Testare General Purpose Processor, ASIP, testare ASIP, Prezentarea finala a proiectului - prezentare, documentatie. Dintre toate milestone-urile doar ASIP si testare ASIP nu au putut fi finalizate.

**Hardware Design**

O arhitectură RISC pe 16 biți ar funcționa pe principiul unui set redus de instrucțiuni, fiecare având o dimensiune de 16 biți. Aceasta înseamnă că fiecare instrucțiune trimisă la procesor ar fi codificată în 16 biți. De exemplu, într-un procesor de 16 biți, adresele de memorie, registrele și majoritatea operațiilor ar fi limitate la o lățime de 16 biți, permițând procesarea datelor în unități de 16 biți la un moment dat.

Componentele principale funcționează astfel:

PC (Program Counter): Deține adresa următoarei instrucțiuni care trebuie executată.

Instr Mem (Memory Instruction): Memoria care stochează instrucțiunile programului.

Instr Reg (Instruction Register): Temporar stochează instrucțiunea curentă care se execută.

Registers: Stochează temporar date pentru procesare rapidă.

ALU (Arithmetic Logic Unit): Efectuează operațiile aritmetice și logice.

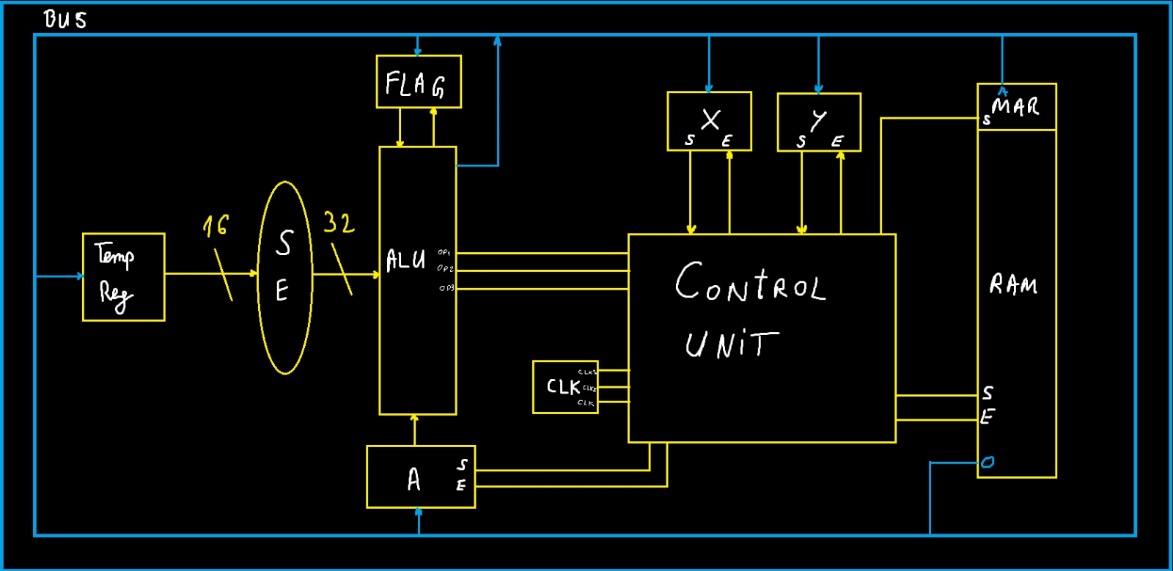
Data Mem (Data Memory): Memoria care stochează datele pe care procesorul le folosește.

Control Unit: Dirijează operațiunile procesorului, inclusiv decodarea instrucțiunilor și coordonarea altor componente pentru a executa aceste instrucțiuni.

Într-un ciclu tipic de execuție, unitatea de control interpretează instrucțiunea curentă, stabilește care trebuie să fie sursele datelor (cum ar fi registrele sau memoria), determină operația necesară în ALU și apoi mută rezultatele unde trebuie să fie stocate, cum ar fi într-un registru sau în memoria de date.

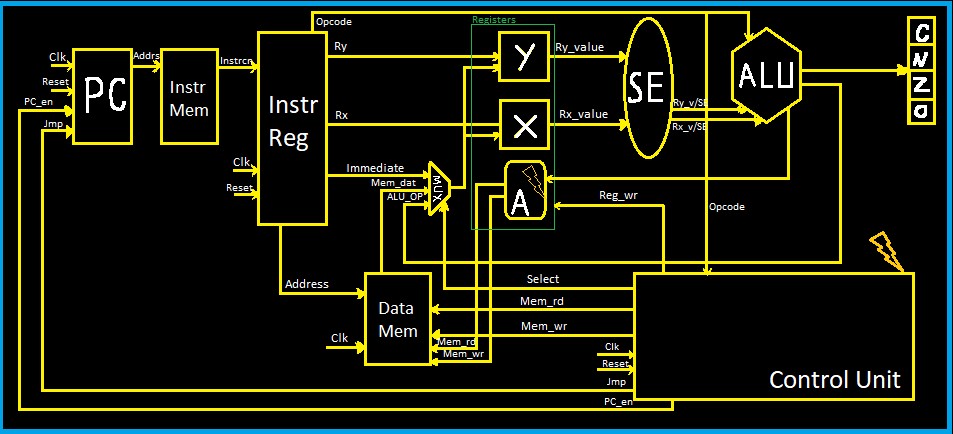
Folosirea unei arhitecturi pe 16 biți influențează performanța, consumul de energie și complexitatea procesorului. Arhitecturile RISC sunt proiectate să fie simple și eficiente, cu instrucțiuni care pot fi executate rapid și care necesită un număr mai mic de cicluri de ceas pentru a fi completate, în comparație cu arhitecturile CISC (Complex Instruction Set Computing). Aceasta poate duce la o performanță mai bună în anumite aplicații și este adesea preferată în sisteme încorporate și hardware cu resurse limitate.

Prima arhitectura a hardware-ului:



Din versiunea initiala lipseau unele componente si erau legate neconform cerintei urmand ca dupa mai multe revizuiri sa se ajunga la varianta finala a arhitecturii.

Arhitectura finala a hardware-ului:



**Software Design**

Arhitectura unui procesor RISC (Reduced Instruction Set Computing) este proiectată să maximizeze performanța prin simplificarea setului de instrucțiuni, concentrându-se pe un număr redus de instrucțiuni simple și eficiente. Această abordare contrastează cu arhitectura CISC (Complex Instruction Set Computing), care include un set mai amplu și complex de instrucțiuni.

Iată câteva caracteristici cheie ale arhitecturii software a unui procesor RISC:

1. Set de Instrucțiuni Redus:

- RISC utilizează un set de instrucțiuni redus pentru a simplifica decodificarea și execuția acestora.

- Instrucțiunile sunt proiectate să fie simple, executând operații de bază, iar sarcinile mai complexe sunt compuse dintr-o serie de instrucțiuni simple.

2. Execuție Pipeline:

- Procesoarele RISC adesea utilizează execuția în pipeline pentru a îmbunătăți performanța.

- Ciclul de execuție al unei instrucțiuni este divizat în mai multe etape care pot fi efectuate în paralel.

3. Acces la Memorie:

- RISC favorizează accesul la memorie prin intermediul unor instrucțiuni simple de încărcare și stocare.

- Adesea, instrucțiunile de manipulare a datelor și cele de acces la memorie sunt separate.

4. Registre în Număr Mai Mare:

- Procesoarele RISC au un număr mai mare de registre în comparație cu cele CISC.

- Aceste registre sunt utilizate eficient pentru stocarea temporară a datelor, reducând astfel necesitatea acceselor la memorie.

5. Predicție de Ramură Simplă:

- Arhitectura RISC se bazează pe strategii simple de predicție a ramurii pentru a minimiza penalizările de întârziere asociate cu instrucțiunile condiționale.

6. Inexistența Instrucțiunilor Complexe:

- Instrucțiunile complexe, cum ar fi cele care acționează direct asupra memoriei, sunt adesea evitate în favoarea unor instrucțiuni simple.

7. Proiectare Modulară:

- Arhitectura RISC favorizează o abordare modulară și uniformă, facilitând dezvoltarea și mentenanța software-ului.

Prin urmare, programarea pentru un procesor RISC implică optimizarea codului pentru a beneficia de caracteristicile specifice ale arhitecturii, cum ar fi utilizarea eficientă a registrelor, gestionarea atentă a acceselor la memorie și maximizarea potențialului de paralelism. Aceste aspecte conduc la o performanță îmbunătățită în anumite scenarii, cum ar fi aplicațiile care necesită execuție rapidă a unui număr mare de instrucțiuni simple.

Odată ce CPU-ul este proiectat și verificat cu succes, software-ul poate fi scris pentru a rula CPU-ul pentru a testa simularea într-un caz potențial de utilizare.

Setul de instructiuni:

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generatedA black rectangular grid with white text

Description automatically generated

Codul sursa pentru ALU:



Codul sursa pentru unitatea de control:





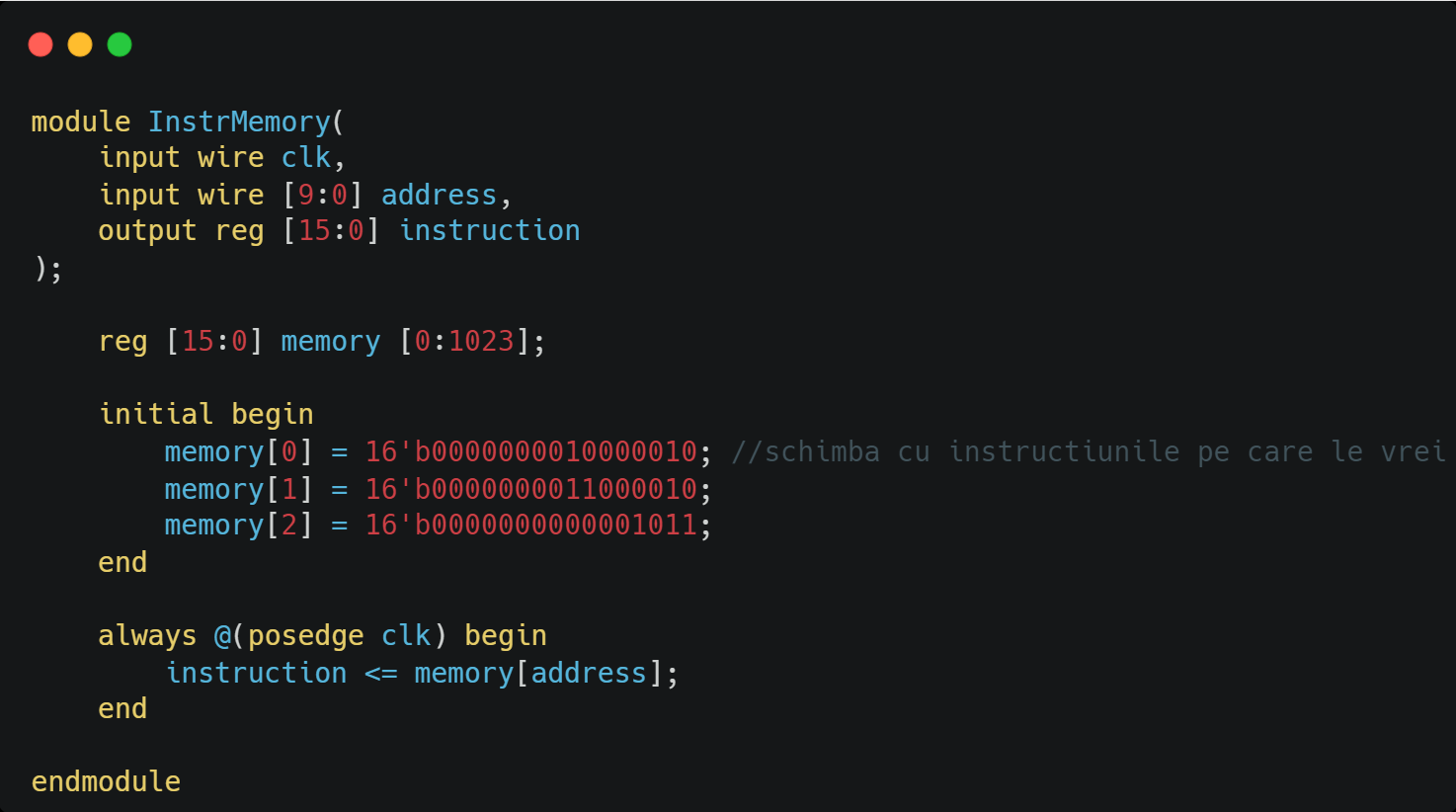
Codul sursa pentru CPU:



Codul sursa pentru Flag Register:



Codul sursa pentru Instruction Memory:



Codul sursa pentru memorie:

A screen shot of a computer program

Description automatically generated

Codul sursa pentru Program Counter:

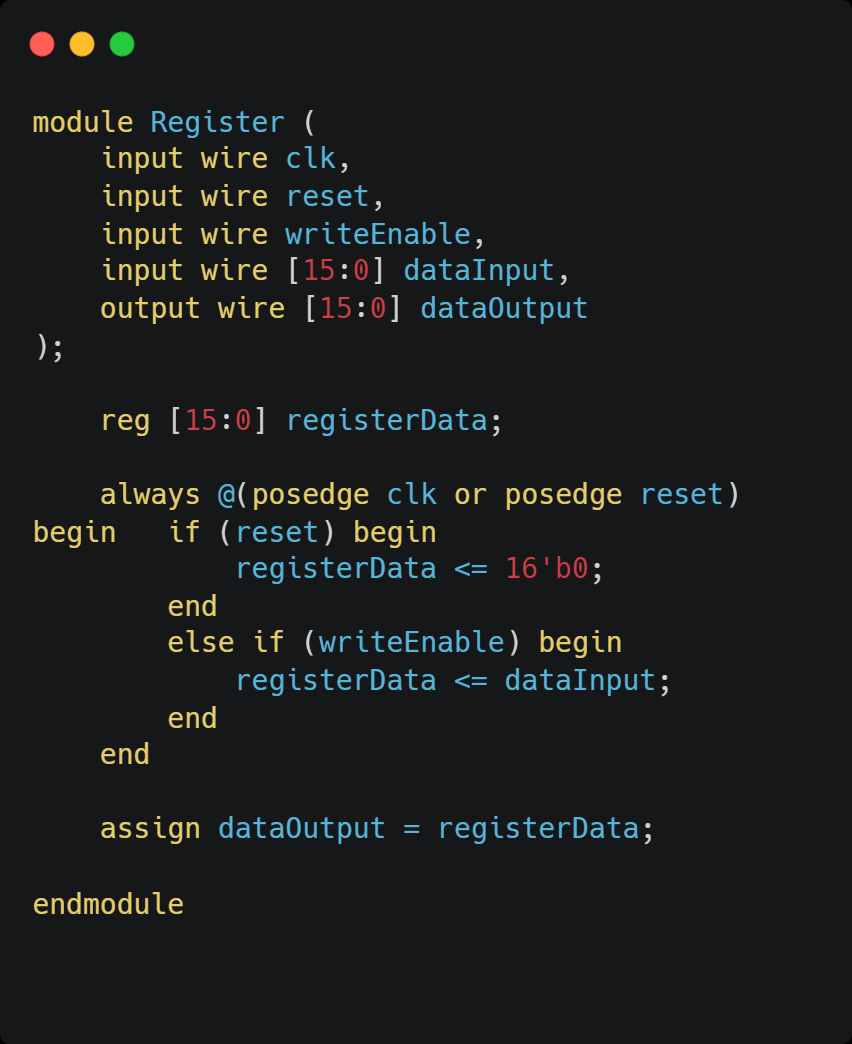
A computer screen with text and numbers

Description automatically generated

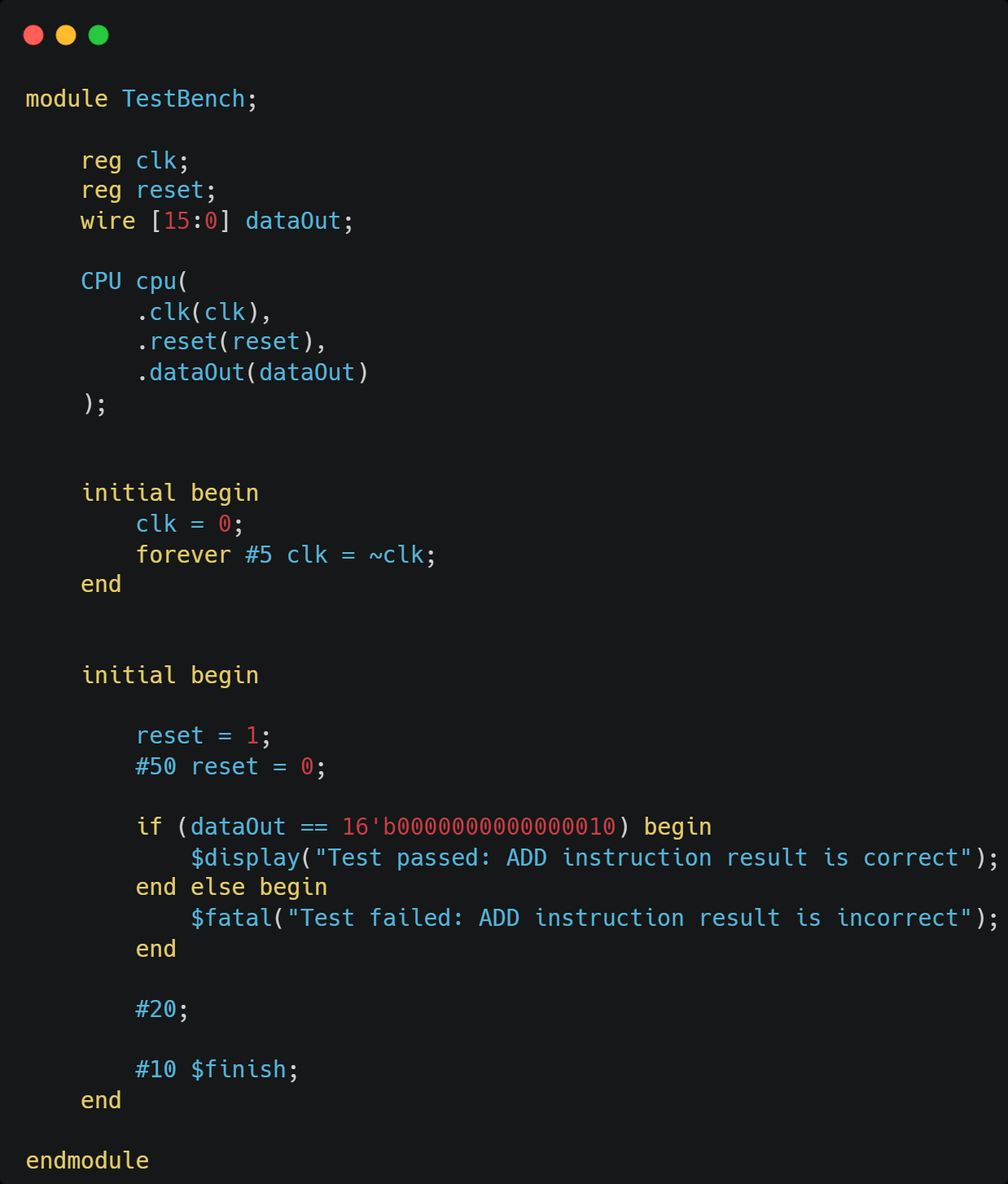
Codul sursa pentru Stack:



Codul sursa pentru Register:



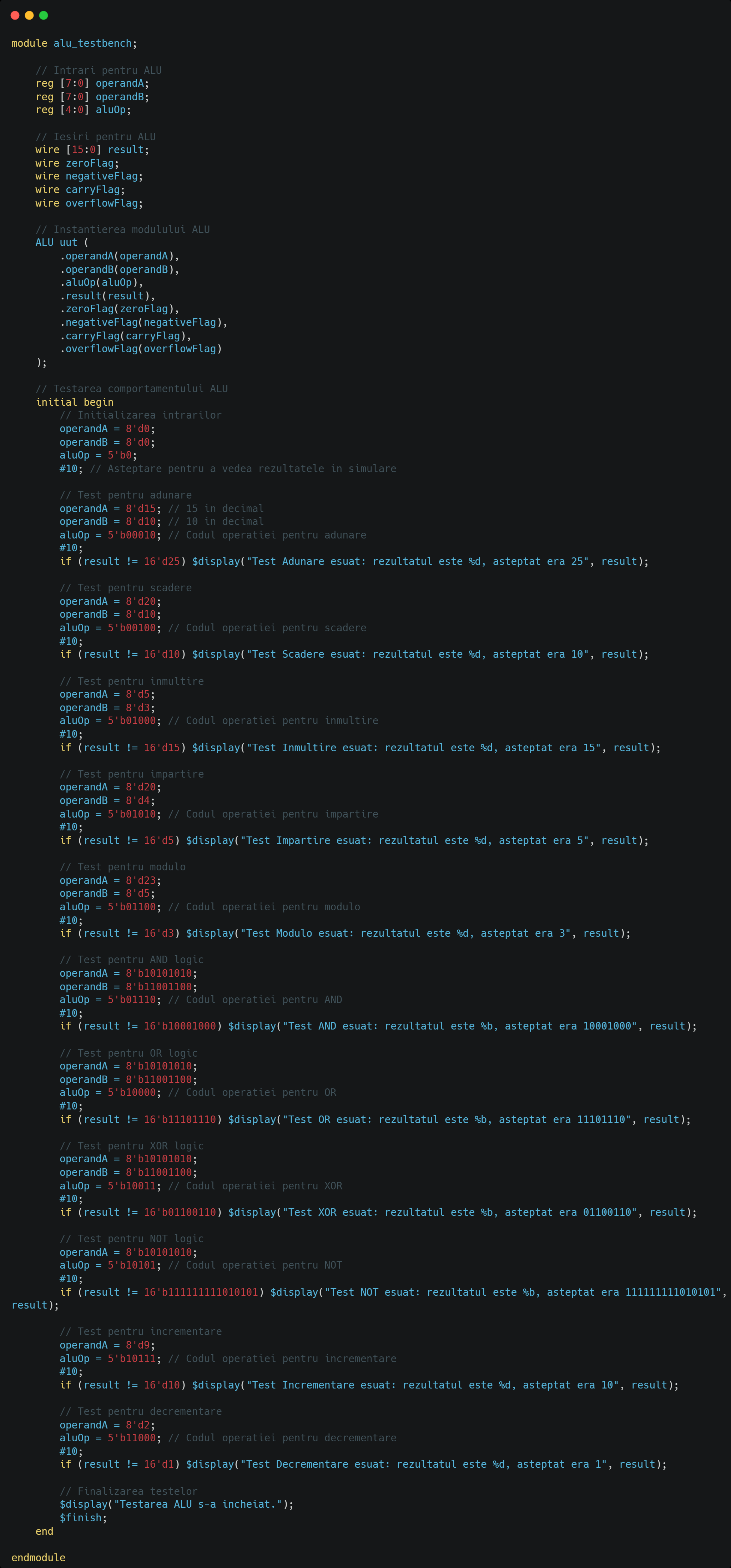
Codul sursa pentru Test Bench:



**Testare**

Pentru modulele ALU, Control Unit, Instruction Memory, Program Counter, Register si Stack au fost facute testbench-uri pentru a le verifica functionalitatea.

Testbench ALU:



Testbench Control Unit:



Testbench Instruction Memory:



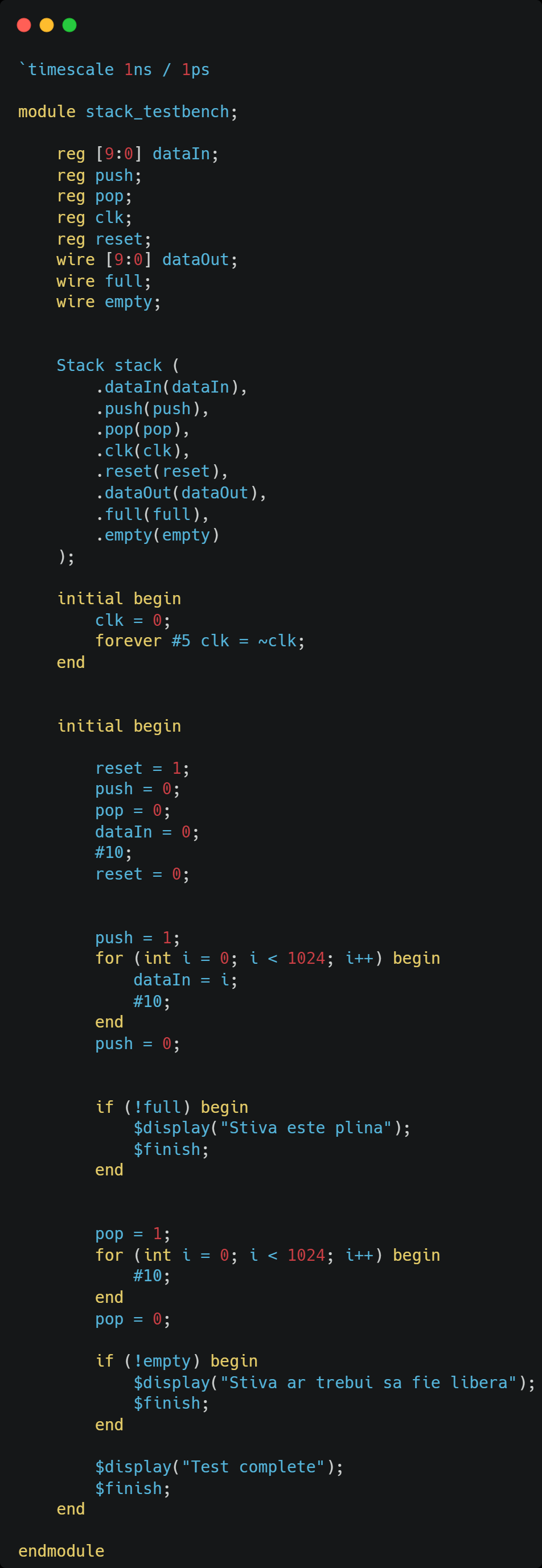
Testbench Program Counter:



Testbench Register:



Testbench Stack:



**Bibliografie**

1. <https://www.quora.com/How-do-I-build-a-processor-16-bit>
2. <https://www.hpcf.upr.edu/~humberto/courses/arch2013/less-broken-cpu.html>
3. https://www.ijert.org/research/design-of-16-bit-risc-processor-IJERTV2IS70828.pdf