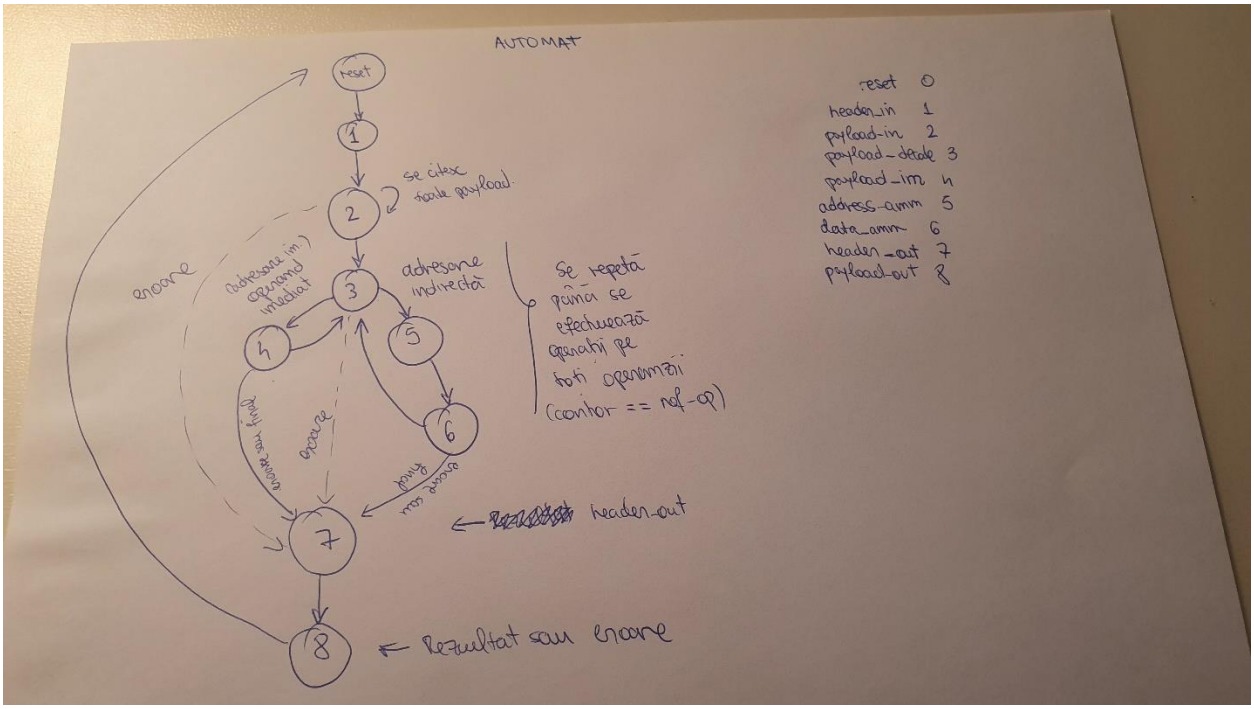
README TEMA 2

Craioveanu Sergiu-Ionut 331AA

SCHITA AUTOMAT:



Implementarea temei a fost realizata cu 8 stări in total. Voi urma prin a descrie funcționalitatea fiecărei stări. Pentru a vedea operațiile efectuate in detaliu, inspectați codul.

0. **Reset**: Sunt intializate toate variabilele (registri) cu 0, in vedere fiind cele folosite intre stari.
1. **Header_in**: Daca bitii valid_in si cmd_in sunt asertati, putem efectua citirea. Stocam codul de operatii si numarul de operanzi.
2. **Payload_in**: Verificam intial pentru erori legate de operanzi, specifice operatiilor. Daca bitul valid_in e asertat, stocam toti operanzii cititi intr-un vector de registri, fiecare element fiind compus din {mod, operand} de adresare. De asemenea, au loc ajustari de clock ce afecteaza indexarea vectorului.
3. **Payload_decode**: Efectuam verificari pentru citirea corecta a clock-ului. Parcurgem vectorul, in functie de tipul de adresare (mod), intram fie in payload_im, fie in address_amm. Altfel, se trimite pachet de eroare.
4. **Payload_im**: Stare pentru efectuarea operatiilor cu modul de adresare direct. Rationament destul de simplu. In functie de codul operatiei, efectuam operatiile cerute. Pentru operatii cu mai multi operanzi, verificam daca ultimul operand citit este cel final (caz in care mergem in header_out) sau daca mai sunt operanzi de citit (unde ne intoarcem in payload_decode).

5. **Address_amm**: Stare cu scopul de a efectua handshake-ul intre memorie si alu. Se aserteaza bitii necesari, impreuna cu adresa, apoi se asteapta amm_waitrequest. Daca bitul este asertat, incepem efectuarea handshake-ului. Daca trec 3 cicli de ceas si totul e in regula, mergem in data_amm pentru a primi datele dorite. Altfel, eroare.
6. **Data_amm**: Daca amm_waitrequest e 0 si rezultatul amm_response e 00, putem incepe efectuarea operatiei, pe care o stocam in result. Altfel, eroare. La fel ca in payload_in, efectuam operatia si verificam daca ultimul operand citit este cel final sau daca mai avem de parcurs din vector.
7. **Header_out**: Numele e destul de sugestiv. Se seteaza bitii necesari pe locurile asignate. Se ajunge in aceasta stare fie prin eroare (imediat), fie prin parcurgerea vectorului si efectuarea operatiilor. Trecem in starea finala.
8. **Payload_out**: Asertam bitul valid_out. In functie de operatie, trunchiam rezultatul. Daca de-a lungul programului bitul de eroare a fost asertat, returnam 0xBAD. Altfel, transmitem rezultatul catre data_out. Trecem in starea de reset, insemnand ca s-a incheiat o parcurgere completa a automatului.