Negoita Andrei Laurentiu
Grupa 332 AA
Tema 2: NOT-SO-SIMPLE ALU

Automatul este format din urmatoarele stari:
-RESET
-IDLE
-FETCH_DATA
-PROCESARE
-OPERATII
-AFISARE

Variabile folosite:
-Pentru memorie am avut nevoie de 2 variabile de tip reg pentru a citi din memorie si pentru a retine adresa amm_read_reg, respective amm_address_reg;
-pentru operatii am folosit opcode pentru a pastra tipul operatiei;
-pentru a vedea ce mod de adresare am (imediata sau indirecta) , am utilizat un vector pentru a retine fiecare mod al fiecarui operand;
-pentru a determina erorile ce pot aparea la unele operatii am utilizat variabila error;
-variabilele result si resultNext au fost folosite pentru a memora rezultatele de la operatii, fara a se rescrie si a-mi permite sa trec la operatia urmatoare;
-vectorul operands l-am utilizat pentru a stoca in el numarul operanzilor pe care ii voi folosi la calcule;
-nr_Operands reprezinta numarul tuturor operanzilor care vor fi indicati in header;
-pentru operatiile din header am utilizat operand_in si operand_in_next pentru a tine cont de operanzii care au intrat in automat si care urmeaza sa fie procesati si calculate;
-pentru payload am folosit operand_proc si operand_proc_next pentru a realiza procesarea operanzilor in functie de modul de adresare;
-pentru realizarea operatiilor, pe langa result si resultNext acolo unde se memora rezultatul, am utilizat operand_cal si operand_cal_next pentru a lua in ordine operanzii ce urmeaza sa fie adaugati la rezultat;

Pentru initializarea tuturor modurilor si operanzilor, fiind vectori, am folosit un for cu un numar finit de iteratii si le-am atribuit tuturor valoarea 0

In always-ul secvential, fiecare variabila primeste prin atribuiri non-blocante valoarea urmatoare a acelei variabile, iar in cazul in care avem reset, urmatoarea stare va fi imediat cea de RESET.

In always-ul combinational, output-urile sunt initializate cu 0 si de asemeni si datele de memorie, urmand ca mai apoi sa inceapa constructia automatului, luand fiecare stare in parte.

Starea de RESET are scopul de a seta toate variabilele la valoarea initiala 0 si se revine in aceasta stare atunci cand se termina de verificat toti operanzii, dupa afisarea rezultatului.

In starea IDLE are loc preluarea datelor dintr-un transfer. Primul cuvant care intra trebuie sa fie validat atat de valid_in cat si de cmd_in care trebuie sa aiba valoarea 1. Aceasta stare este asociata headerului. Daca se respecta conditiile, atunci se va memora codul operatiei si se va retine numarul de operanzi.

In plus, tot in aceasta stare, tratram eventualele erori ce pot sa apara la unele operatii. Stiind numarul de operanzi, in cazul in care acesta este 0, vom returna eroare si vom merge direct in stare de afisare.

Daca numarul de operanzi este diferit de 1, atunci operatiile NOT, INC, DEC si NEG nu se pot efectua, deoarece acestea sunt valabile doar pentru un operand. De asemeni, daca numarul de operanzi este diferit de 2, atunci nu vom putea efectua operatiile de shiftare SHL si SHR.

In toate aceste cazuri vom avea eroare si vom merge direct in starea de afisare, altfel urmeaza starea de FETCH_DATA.

In starea de FETCH_DATA, avand valid_in egal cu 1, putem incepe aducerea de informatii din payload. Astfel, din datele de intrare, preluam modul de adresare si operandul/adresa pentru fiecare operand in parte. Toate acestea le realizam prin intermediul vectorilor si realizam acest proces pana avem datele pentru toti operanzii.

In schimb, daca valid_in nu este 1, dar numarul de operanzi este egal cu numarul de operanzi intrati deja, putem trece la starea de PROCESARE.

In starea de PROCESARE, examinam modurile de adresare si in functie cum sunt acestea mergem in starea de OPERATII, daca avem adresare imediata, sau in starea de PROCESARE+1, daca avem adresare indirecta si trebuie sa lucram cu zona de memoria.

Astfel, daca modul este 2’b01, asertam amm_read_reg si amm_address_reg primeste adresa de la operand, care a fost atribuita in starea precedenta, in FETCH_DATA, urmand ca sa trecem in starea de PROCESARE+1.

Daca nu avem acest mod, incrementam operandul procesat si revenim tot in aceasta stare.

Starea de PROCESARE+1 este dedicata lucrarii cu zona de memorie. Asfel, daca amm_waitrequest este 0 si amm_response este tot 0, putem pune in vectorul de operanzi data citita din memorie. Dupa acest lucru, trecem la urmatorul element din vectorul de operanzi si ne intoarcem in starea de PROCESARE si reluam procesul.

In schimb, daca amm_wairequest este egal cu 0 si amm_response diferit de 0, vom avea parte eroare si vom merge direct in starea de AFISARE. Daca aceste conditii nu sunt indeplinite, inseamna ca revenim tot la zona de memorie. Pentru a incepe din nou lucrul cu zona de memorie trebuie sa asertam amm_read_reg si de asemeni sa oferim adresa de memorie.

Dupa ce observam modul de adresare, ajungem la zona de OPERATII. In aceasta stare verificam daca operanzii calculati sunt egali cu numarul total de operanzi. Daca acest lucru se intampla ajungem in zona de AFISARE. Insa pentru a ajunge in aceasta situatie trebuie sa efectuam operatii, iar la fiecare operatie sa incrementam numarul de operanzi calculati.

Pentru efectuarea operatiilor, ne folosim de variabilele result si resultNext in care stocam rezultatul fiecarei operatii, fara a deteriora rezultatul precedent.

Dintre toate operatiile, singurele unde exista mici detalii de explicat sunt la:

-AND: unde in cazul in care avem un singur operand trebuie sa luam numai acea valoarea, fara a-i adduce modificari; in cazul in care sunt mai multi operanzi, se realizeaza “si”-ul intre acestia

-SHR/SHL: deoarece avem nevoie de 2 operanzi pentru aceste operatii, la incrementarea operand_cal_next trebuie sa adunam un 2.

Pentru toate operatiile la care se utilizeaza un operand sau doi, nu se va mai reveni in aceasta stare de OPERATII. Erorile cu privire la numarul de prea mare de operanzi pentru anumite operatii au fost tratate in starea IDLE.

Starea de AFISARE are rolul de a afisa Headerul. Pentru acest lucru trebuie ca valid_out si cmd_out sa fie asertate. Data de iesire va fi formata din 11 biti de 0, eroare si codul operatie. Dupa aceasta se trece in starea de AFISARE+1 care corespunde payloadului.

Pentru payload, in AFISARE+1, trebuie sa avem doar valid_out asertat. Tot in payload verificam si eventualele erori care ar fi putut sa apara pe parcursul tuturor starilor. In cazul in care avem error egal cu 1, vom afisare mesajul de eroare BAD.

In cazul in care nu avem erori, afisam payloadul, folosind result, cel pe care l-am utilizat pentru calculul operatiilor.

Dupa ce se afizeaza datele de iesire, putem incepe un nou ciclu, iar pentru asta trebuie sa ajungem in RESET.