DOCUMENTATIE

MIPS PIPLINE

NUME STUDENT: Pântea Marius-Nicușor

GRUPA:30222

**Introducere**

Acest document servește drept specificație a designului în limbajul VHDL pentru un procesor pipeline. Designul implementează un procesor cu mai multe etape, inclusiv Fetch de Instrucțiuni (IF), Decodare de Instrucțiuni (ID), Executare (EX), Memorie (MEM) și Scriere înapoi (WB). Scopul designului este de a realiza un procesor capabil să execute instrucțiuni într-un mod eficient și corect conform arhitecturii specificate.

**Elemente funcționale:**

1.Fetch de Instrucțiuni (IF)

Obiectiv: Extrage instrucțiunile din memoria de program și le furnizează către etapa următoare a pipeline-ului.

Funcționalități:

* Ciclul Fetch: La fiecare ciclu de ceas, adresa curentă a programului este folosită pentru a citi instrucțiunea din memoria de program.
* Incrementare PC: Adresa programului (PC) este incrementată pentru a se pregăti pentru citirea următoarei instrucțiuni.
* Transmiterea Instrucțiunilor: Instrucțiunea citită este transmisă etapei de Decodare de Instrucțiuni (ID) pentru a fi procesată.

2.Decodare de Instrucțiuni (ID)

Obiectiv: Decodează instrucțiunile extrase și pregătește operațiunile necesare pentru etapa de execuție.

Funcționalități:

* Decodarea Tipului de Instrucțiune: Instrucțiunile sunt decodate pentru a determina tipul (aritmetică, de control, de acces la memorie etc.) și operațiunile asociate.
* Extragerea Operanzilor: Identifică și extrage operanzilor necesare pentru execuție, cum ar fi registrele sursă și destinație sau adresele de memorie.
* Detectarea Dependințelor de Date: Verifică dacă instrucțiunea curentă depinde de rezultatele altor instrucțiuni și controlează execuția în consecință pentru a evita hazardurile de date.

3.Executare (EX)

Obiectiv: Execută operațiunile specifice instrucțiunilor într-un mod eficient și corect.

Funcționalități:

* ALU Operations: Pentru instrucțiunile aritmetice și logice, se utilizează o unitate aritmetică-logică (ALU) pentru a efectua operațiile specificate.
* Control Flow Operations: Pentru instrucțiunile de control, cum ar fi săriturile și ramificările, se determină adresa următoare a programului în funcție de condiția specificată.
* Accessing Memory: Pentru instrucțiunile care implică accesul la memorie (citire sau scriere), se efectuează accesurile corespunzătoare și se obțin/scriu datele necesare.

4.Memorie (MEM)

Obiectiv: Accesează memoria de date pentru citire sau scriere, în funcție de necesitățile instrucțiunilor.

Funcționalități:

* Read/Write Operations: Accesează memoria pentru a citi sau scrie date, respectând adresele specificate de instrucțiuni.
* Data Synchronization: Asigură sincronizarea corectă a acceselor la memorie pentru a evita conflictul și pentru a garanta coerența datelor.
* Data Forwarding: Dacă este necesar, furnizează datele citite sau calculate etapelor ulterioare ale pipeline-ului.

2.5 Scriere înapoi (WB)

Obiectiv: Actualizează registrele sau memoria, dacă este necesar, cu rezultatele finale ale execuției instrucțiunilor.

Funcționalități:

* Updating Registers/Memory: Actualizează registrele de date sau memoria de date, în funcție de tipul instrucțiunii și de rezultatele obținute.
* Controlul Scrierii înapoi: Gestionează scrierea în registre sau memorie în conformitate cu politicile de acces și cu semnalizările specifice ale instrucțiunilor.

**Elemente nefuncționale:**

1 Performanță

Obiectiv: Asigurarea unei execuții eficiente a instrucțiunilor, minimizând întârzierile și optimizând utilizarea resurselor.

Cerințe:

* Timpul de execuție al unei instrucțiuni să fie cât mai mic posibil.

3.3 Extensibilitate

Obiectiv: Permitea extinderii și adaptării ușoare a designului pentru a suporta noi instrucțiuni sau caracteristici.

Cerințe:

* Design modular și flexibil, care să permită adăugarea sau modificarea funcționalităților cu efort minim.
* Documentare adecvată și comentarii în cod pentru a facilita înțelegerea și modificarea ulterioară a designului.

**Precizări:**

* Toate elementele componentele au fost testate pe placa.
* Diferențele fata de Mips-ul standard sunt adăugarea semnalelor necesare pentru instrucțiunea branch grater than zero , celelalte instrucțiuni modifica ALU (Adaugand semnalele in UC si EX necesare)

**Probleme întâmpinate**

* Erori în interconectarea modulelor sau a semnalelor
* Erori la numărul de biți al registrelor
* Necorespunderea semnalelor între module
* Erori în procesul de scriere sau citire a datelor în/din registre

**Cele 4 instrucțiuni adăugate:**

XOR – bitwise eXclusive-OR

* SAU-Exclusiv logic între două registre (sursa si target), memorează rezultatul in registrul destinație si incrementează program counter-ul
* $d 🡨 $s ^ $t; PC 🡨 PC + 4;
* xor $d, $s, $t
* 000000 sssss ttttt ddddd 00000 100110

SLT – Set on Less Than (signed)

* Instrucțiunea SLT compară două registre (sursa și target) și setează un al treilea registru (destinatie) la valoarea 1 dacă $s este mai mic decât $t și la valoarea 0 în caz contrar
* PC🡨PC + 4; if $s < $t then $d 🡨 1 else $d 🡨 0;
* slt $d, $s, $t
* 000000 sssss ttttt ddddd 00000 101010

ORI – bitwise OR Immediate

* Valoarea din registrul sursa este combinată folosind operația logică SAU cu o valoare imediată iar rezultatul este stocat în registrul $t, în timp ce program counter este incrementat pentru a trece la următoarea instrucțiune din cod în alt registru
* $t 🡨$s | ZE(imm); PC 🡨PC + 4;
* ori $t, $s, imm
* 001101 sssss ttttt iiiiiiiiiiiiiiii

BGTZ – Branch on Greater Than Zero

* Verifică dacă valoarea din registrul sursa este strict mai mare ca 0, iar daca da aduna la program counter offset-ul, echivalent cu numărul de instrucțiuni peste care să sară. Daca nu incrementează program counter-ul.
* If $s > 0 then PC PC + 4 + (SE(offset) << 2) else PC PC + 4;
* bgtz $s, offset
* 000111 sssss 00000 oooooooooooooooo