**FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE**

**DEPARTAMENTUL CALCULATOARE**

**PROIECT**

La disciplina

**ARHITECTURA CALCULATOARELOR**

**Profesor coordonator: Nume student:**

**Muresan Mircea Paul Pal Tudor Ovidiu**

**Grupa: 30224, semestrul II, 2022**

### An academic: 2021 – 2022

**Cuprins**

### Introducere

### Instructiuni MIPS

### Tabel cu valorile variabilelor de control

### Explicatie cod

### Corectitudinea codului

### Bibliografie

1. **Introducere**

Proiectul presupune dezvoltarea unui procesor MIPS, ciclu unic, pe 16 biți folosind limbajul de descriere hardware VHDL si Vivado pentru a putea testa microprocesorul pe plăcută FPGA. Programul meu parcurge un sir si afișează 1 daca maximul si minimul din sir sunt pare, sau 0 in caz contrar.

1. **Instructiuni MIPS**

Pentru MIPS putem sa avem 3 tipuri de instrucțiuni:

* De tip R
* De tip I
* De tip J

Din setul minimal de instrucțiuni al procesorului fac parte urmatoarele:

De tip R:

* ADD (adăugare)
* SUB (scădere)
* SLL (shift left logic)
* SRL (shift right logic)
* AND
* OR

De tip I:

* ADDI (add immediate)
* BEQ (branch on equal)
* LW (load word)
* SW (store word)

De tip J:

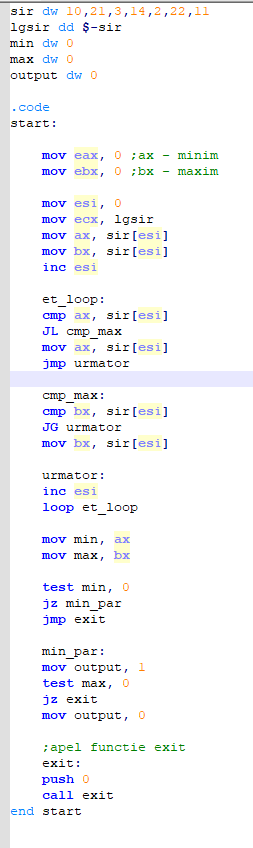
* Jump

Pe lângă aceste instrucțiuni, am mai adăugat doua instrucțiuni de tip R: XOR si SGT (set on greater than) si doua instrucțiuni de tip I: BGTZ (Branch on Greater Than Zero) si BLTZ (Branch on Less Than Zero) pentru a putea verifica mai ușor minimul si maximul din sir.

1. **Tabel cu valorile variabilelor de control**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Instrucțiune** | **Opcode** *Instr(15-13)* | **RegDst** | **ExtOp** | **ALUSrc** | **Branch** | **J** | **MemWrite** | **MemtoReg** | **Reg Write** | **ALUOp (2:0)** | **func**  *Instr(2-0)* | **ALUCtrl (2:0)** |
| ADD | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 000 | 000 |
| SUB | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 001 | 001 |
| SLL | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 010 | 010 |
| SRL | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 011 | 011 |
| AND | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 100 | 100 |
| OR | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 101 | 101 |
| XOR | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 110 | 110 |
| SGT | 000 | 1 | X | 0 | 0 | 0 | 0 | 0 | 1 | 000 | 111 | 111 |
| ADDI | 001 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 001 | - | 000 |
| BEQ | 010 | X | 1 | 0 | 1 | 0 | 0 | X | 0 | 010 | - | 001 |
| BGTZ | 011 | X | 1 | 0 | 1 | 0 | 0 | X | 0 | 011 | - | 010 |
| BLTZ | 100 | X | 1 | 0 | 1 | 0 | 0 | X | 0 | 100 | - | 011 |
| LW | 101 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 101 | - | 100 |
| SW | 110 | X | 1 | 1 | 0 | 0 | 1 | X | 0 | 110 | - | 101 |
| J | 111 | X | X | X | X | 1 | 0 | X | 1 | 000 | - | - |
|  |  |  |  |  |  |  |  |  |  |  |  |  |

1. **Explicatie cod**

O imagine care conține text, tablă albă

Descriere generată automat

O imagine care conține text

Descriere generată automat

Am atașat o poza cu codul in C si cu traducerea in cod mașina. Voi lucra cu 7 registre (n, adresa, count, maxim, minim, a, aux) si unul pentru 0. Am decis sa inițializez fiecare registru din memory block, folosind Load Word, chiar daca registrele au niște valori inițiale. N fiind numărul de elemente a șirului, acesta se inițializează cu prima valoare din memoria RAM. In A vor fi salvate si verificate elementele de pe pozițiile 1->n din memoria RAM (in cazul nostru pozițiile 1, 2, 3, 4). Pentru a putea folosi Load Word pentru A avem nevoie de un registru adresa care reprezintă poziția de la care se face load word pentru fiecare A. Count se inițializează cu 1 si la fiecare loop, adresa <= adresa + count.

In fiecare iterație a loop-ului, se verifica daca a(adresa) este mai mare decât maxim. In caz ca e mai mare, se face branch peste instrucțiunea de jump (offset 1) si maxim <= a(adresa) prin intermediul Store Word si Load Word. Daca nu este mai mare, se verifica daca a(adresa) < minim. In caz ca e mai mic, se face branch peste instrucțiunea de jump si minim <= a(adresa) prin intermediul Store Word si Load Word. Daca este mai mic, se face jump înapoi la instrucțiunea de incrementare a adresei.

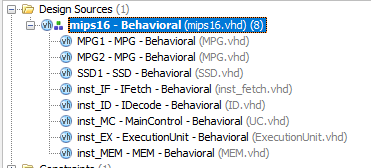
După bucla, se verifica daca ultimul bit al minimului este 0. Daca acesta nu este 0, se face jump la final unde a devine 0 (minim/maxim nu este par). Daca acesta este 0, înseamnă ca numărul este par, si se face branch peste jump, apoi se verifica daca ultimul bit al maximului este 0. Daca maximul este impar, a <= 0 prin LW a, 7. Daca max e par, se sare peste aceasta instrucțiune si se stochează valoarea lui a pe poziția 5 a memoriei RAM.

Tabel de adevăr pentru ultimul bit de la maxim, minim si pentru a:

|  |  |  |
| --- | --- | --- |
| Maxim | Minim | A |
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

1. **Corectitudinea codului**

Procesorul contine toate componentele necesare descrise in VHDL (Instruction Fetch, Instruction Decode, Seven segment display, doua MPG, Main Control, Execution Unit si Memory block-ul)

.

Programul compileaza si genereaza bitsream-ul fara probleme, dar nu am reusit sa verific corectitudinea pe placuta FPGA, pentru ca proiectul nu era gata miercuri cand aveam sansa sa il pun pe placuta.

1. **Bibliografie**

<https://users.utcluj.ro/~onigaf/files/AC.html>

<https://www.youtube.com/channel/UCaw7etQgv0cfDcDdnBAnCBw>