

- mips\_instr\_mem modülü
- control modülü
- mips\_registers modülü

- extend modülü
- mux\_32bit modülü
- alu modülü
- mips\_data\_mem modülü
- mux\_32bit modülü
- mux\_5bit modülü
- mips\_registers modülü
- mux\_PC modülü

### 1.3

Jump ederken PC'yi değiştirmek için kullandığımız  $PC = \{ \{PC+4[31:28]\}, \text{jump\_address}, 2'b0 \}$ ; sürekli hata verdi bu sebeple jump'la ilgili olan her şeyi silmek zorunda kaldım.

ModelSim'de bazı outputlarda x çıkıyor. Bunu da düzeltemedim.

## 2. Metod

Instruction fetch edildikten sonra opcode'u control'e gider ve opcode'a göre datapath'te yapılacak işlem belirlenir. Instruction'ın kalan kısımları da fetch edildikten sonra hangi register'daki değer kullanacağını anlamak için ve sign extend edilemek için kullanılır. İstenen register'daki datalar (ya da sign extend edilmiş data) gerekli işlemi yapmak üzere ALU modülüne gönderilir. Orada gerekli işlem yapıldıktan sonra eğer branch edilmesi gerekiyorsa branch yapılır, memory'yle ilgili bir işlem varsa Result oraya gönderilir, bunların ikisi de yoksa write back yapmak için register modülüne gönderilir.

En sonunda bulunan sonuç ilgili register'a yazılır ya da istenen memory adresine yazılır ya da jump işlemi yapılır. PC'nin yeni değerinin gösterdiği yerdeki instruction alınarak aynı döngü program bitene kadar tekrarlanır.

### 2.1

#### 1) mips\_instr\_mem (instruction, PC)

**input** = PC

**output** = instruction

PC'nin değerine göre istenen instruction'ı verir.

#### 2) control (opcode, sig\_reg\_dest, signal\_reg\_write, sig\_branch, sig\_mem\_read, sig\_mem\_write, sig\_alu, sig\_memToReg, sig\_alu\_src, alu\_op)

**input** = opcode

**output** = sig\_reg\_dest, signal\_reg\_write, sig\_branch, sig\_mem\_read, sig\_mem\_write, sig\_alu, sig\_memToReg, sig\_alu\_src, alu\_op

Gelen opcode'a göre datapath'ın sitenen işlemi yapması için gereken sinyalleri üretir.

#### 3) mips\_registers

(read\_data\_1, temp, write\_data, read\_reg\_1, read\_reg\_2, write\_reg, signal\_reg\_write, clock, zero)

**output** = read\_data\_1, read\_data\_2

**input** = write\_data, read\_reg\_1, read\_reg\_2, write\_reg, signal\_reg\_write, clock, zero

Gelen sinyallere göre gelen register'daki değeri verir ya da gelen veriyi istenen register'a yazar.

#### 4) mux\_32bit (read\_data\_2, sign\_ex, temp, sig\_alu\_src)

**output** = out,

**input** = in1, in2, signal

Gelen sinyale göre in1 ya da in2'yi output olarak verir.

#### 5) alu (read\_data\_1, read\_data\_2, opcode, shamt, zero, result)

**output** = result, zero, sig\_disable

**input** = read\_data\_1, read\_data\_2, opcode, shamt, resul, zero, sig\_disable;

Gelen opcode'a göre gelen datalar üzerinde işlem yapar ve result olarak output verir.

Branch durumunda zero sinyalini 1 yapar.

#### 6) mips\_data\_mem (read\_data, result, write\_data, sig\_mem\_read, sig\_mem\_write, zero)

**output** = read\_data

**input** = result, write\_data, sig\_mem\_read, sig\_mem\_write, zero

sig\_mem\_read ve sig\_mem\_write sinyallerine göre okuma ya da yazma yapar okunan değer olarak read\_data'yı verir, yazılacak değer için write\_data'yı ve adresi alır.

#### 7) mux\_5bit (write\_reg , read\_reg\_2, rd, sig\_reg\_dest)

**output** = write\_reg

**input** = read\_reg\_2, rd, sig\_reg\_dest)

Gelen sinyale göre read\_reg\_2 ya da rd'yi output olarak verir.

#### 8) mux\_PC (temp2, temp, sign\_ex, zero, sig\_branch, clock);

**output** = PC,

**input** = temp, sign\_ex, zero, sig\_branch, clock, sig\_disable

Normal durumda PC'yi 1 artırır

Zero 1'ken yani branch durumunda sign extend'i 2 kere shift edip değeri PC'ye atar.

#### 9) extend(out, in)

**input** = in

**output** = out

Gelen sayıyı 16 bitten 32 bite extend eder.

## 3. Sonuç

### 3.1

