Computer Organization Final Project

Instruction:

Life cycle of 1 instruction:

İnstruction.mem dosyasından okunan instructionlar 2 boyutlu memory arrayinde tutulur ve benim read_index olarak tuttuğum program counter işlemine göre read_indexin belirlendiği değişken arrayden hangi instructionı memoryden almam gerektiğini gösterir.İnstruction alındıktan sonra mip_core modülünde parçalara ayrılır.Gelen opcodelara göre işlem sonucu değerin registera mı yoksa memorye mi yazılacağı belirlenir.Yani memWrite a 0 mı 1 mi atanacağı belirlenir. mips_registers modulunde dönen 2 registerın değerleri, ya da bir register ve bir immediate ile işlem yapan bir insturction ise 1 register ve bir immediate degeri Aluya yollanır.Bu arada ALUya yollanmadan önce gelen parametre değerlerinden biri immediate değer ise 32 bite extend edilirler. Ben ALU modülüm içerisinde Add, addi ,addi,and,andi ,beq ,bne,lui ,or ,ori ,slt ,slti,sltiu ,sll ,sllv ,sra,srl işlemlerinin gerçekleştiği modüllerimi çağırdım ve her birinden çıkan sonucu farklı result değişkenleri içerisinde tuttum ve gelen instruction r type ise func koduna eger gelen fonksiyon I type ise opcoduna bakarak çalışan doğru fonksiyonu belirleyip o fonksiyondan çıkan sonuç değerini doğru sonuç degeri olarak tuttum.Daha sonra memWrite 1 ise mips__data_memory modülüne girer ve çıkan result değerim memorynin hangi adresine yazma işlemi yapılacağını gösterir.Ve o adrese yazar.Yine aynı şekilde eğer instructionım memoryden okuma yapan bir instrucition ise gelen result değerim memorynin hangi adresindeki verinin çekileceğini işaret eder ve oradaki veri alınır,döndürülür.Daha sonra akışta PC bölümü yer alır.Yani program counter değerinin belirlendiği bölüm program counter gelen instructiona göre 4 artar(Bu bizim +1 yapmamızı gerektiriyor çünkü arrayde tuttuğumuz için),+4+extenImmediate kadar artar ya da jump instructionı geldiğinde instruction[25:0] kısmında bulunan değerin extend edilmiş kısmı kadar artar.Bulunan değer benim artık bir sonraki almam gereken instruction bu program counter ile belirlenen değerin gösterdiği instructiondır.Program instructionlarım bitene kadar tek tek bu aşamalardan geçerek son bulur.

32-bit MIPS processor:

module mips_core(clock);

1) sign_extend singEx(extImm, extJump, immediate, jumpAdress);

2) mips_registers registers(register1,register2,writeEnable,rs,rt,writeData);

3) mips_alu aluModule(result, carryOut, temp, shamt, register1, var1);

4) mips_data_memory data_mem(readData, result, register2, memWrite);

5) pc PCreg(PC , readI, clock);

6) mips_instruction_memory Instruction(readI , instruction);

Module mips_core içerisinde 6 tane modül çağırır.

mips_core fetch decode execute işlemlerinin toplandığı modüldür.

mips_core programımızın top-level entitysidir.

Yukarıdaki şekilde ana modülüm yani tüm akışın sağlandığı mips_core modülünün içerisinde çağrılan diğer modüller sırası ile gösterilmiştir.mips_core modülü öncelikle gelen instructionları parçalara ayırır:

```
//resolution by instruction
        assign  opCode=instruction[31:26];
        assign  rs= instruction[25:21];
        assign  rt= instruction[20:16];
        assign  rd= instruction[15:11];
        assign  shamt= instruction[10:6];
        assign  funct= instruction[5:0];
        assign  immediate= instruction[15:0];
        assign  jumpAdress=instruction[25:0];
```

METHODS :

1) sign_extend singEx(extImm, extJump, immediate, jumpAdress);

Bu module eğer immediate değeri ile işlem yapan bir instruction gelirse instructionımızın [15:0] lık kısmında bulunan immadiate değeri parametreye gelir ve bu module içerisinde 32 bite extend olur.
Eger gelen instructionımız jump ise bu bize instructionımızın [25:0]lık kısmındaki jumpAdress par çasına sahip olduğumuzu gösterir ve bu kez jumpAdress kısmı 32bite extend edilir.

sign_extend.v

```
        module sign_extend(extendedI, extendedJ, immediate, jumpAdress);
        output [31:0] extendedI, extendedJ;
        input [25:0] jumpAdress;
        input [15:0] immediate;

        //extended version of immediate part
        assign extendedI = { {16{immediate[15]}}, immediate[15:0]};

        //extended version of jump part
        assign extendedJ = { {6{jumpAdress[25]}}, jumpAdress[25:0]};
```

2) mips_registers registers(register1,register2,writeEnable,rs,rt,writeData);

Mips_registers modülünde 32 tane 32bitlik registerlar dosyadan okunarak regData arrayinin içerisinde tutulur.ve gelen parametredeki adreslerde bulununan register değerleri döndürülür ve eger memWrite 1 gelirse gelen adresteki registerın değerinin değişeceğini gösterir

mips_registers.v

```
module mips_registers(DataRs, DataRt, memWrite, rs_adress, rt_adress, writeData);
output wire[31:0] DataRs, DataRt; // content of rs and rt
input [4:0] rs_adress, rt_adress; // adress of rs and rt
input [31:0] writeData;
input memWrite;
reg [31:0] regData [31:0];
initial begin
        $readmemb(".\\registers.mem", regData) ;
end
assign DataRs = regData[rs_adress];
assign DataRt = regData[rt_adress];
always @(writeData)
        if(memWrite == 1'b1 )
                begin
                        regData[rt_adress] = writeData;
                end
endmodule
```

3) mips_alu aluModule(result, carryOut, temp, shamt, register1, var1);

Buradaki var1 degeri gelen instrructiona göre register2 mi olacak yoksa immediate bir değer mi olacak opcode koşullarına göre değişiklik gösteriyor.Bu koşulu da aşağıdaki şekilde sağlıyorum.

assign var1 = (opCode == 6'b000100) ? register2 : ((opCode == 6'b000101) ? register2 : (opCode == 6'b0) ? register2 : extImm) ;

Ben bu modül içerisinde add,addi,addi,and,andi,bne,lui,_or,slt,slti,sltiu,sll,Sllv,sra,srl, sub işlemlerini gerçekleştiren moduülleri çağırdım ve var1e göre instructionın hangi işlemi gerçekleştiriyor ise o işlemden dönen result değerini asıl sonuç değerim olarak tuttum.ve onu geri döndürdüm.

Buradaki temp parametresine mips_coreda belirlenen gelen instruction r type ise func değeri, I type ise opcode u atanıyor.

4) mips_data_memory data_mem(readData, result, register2, memWrite);

Bu parametredeki memWrite'a mips_core modülünde,gelen instructionın opcodeuna göre bazı I type instructionlarda(Aludan çıkan result değerin memorye yazılması gereken instructionlarda) 1 değeri atandı. Aluda hesaplanan result memorynin hangi adresine yazma ya da hangi adresinden okuma yapılacağını belirtiyor.gelen register değeri hesaplanan adrese yazılır ya da o adresten alınan değer döndrülür.

mips_data_memory.v

```
module mips_data_memory (data_read, add, data_write, mem_write);
        output [31:0] data_read; //output of load word
        input mem_write; //signal that allows writing
        input [31:0] add, data_write ;
 reg [31:0] memory[255:0];
        reg [31:0] data_read;
        initial begin
                $readmemb(".\\data.mem", memory) ;
        end
        always @(add or data_write or data_read)  //If one of these changes
                if (mem_write)
                        memory[add] = data_write;
                else
                        data_read = memory[add];
endmodule
```

5) pc PCreg(PC , readI, clock);

Bu modülü çağırmadan önce PC değerini şu şekilde belirliyorum:

```
assign var2 = readI + 1;// adding 1 because of using array
assign var3 = var2 + extImm;
assign temp4 = var2 + extJump; // jump
 //choosing PC whether to instruction branch or not
assign PC = (opCode == 6'b000100 ) ? (result ? var3 : var2 ):((opCode == 6'b000101 ) ? (result ? var3
: var2 ):(opCode == 6'b000010) ? temp4 : var2 );
```

Yukarıda göstermek istediğim Program counterın değer değişikliği gelen instructionlara göre farklılık gösteriyor.Mesela beq instructionı gelirse ve 1 sonucunu döndürürse hem program counterı 4 arttırırız (+1 olarak göstermemizin sebebi instructionları tutarken array kullanmamız) daha sonra extend edilmiş ExtImm ile toplanır.Load,addu,store,ori gibi instructionlar için sadece +1 yaparız.Daha sonra güncellenen PC değeri pc PCreg(PC , readI, clock) fonksiyonuna yollanır.

pc.v

```
pc(addr , readI, clock);
        output reg [31:0] readI;
        input [31:0] addr;
        input clock;
        reg [31:0] tempPc;
        initial tempPc = 0;

        always @(addr or tempPc)
                begin
                        tempPc = addr;
                end
        always @(clock or tempPc)// running the program with this clock
                begin
                        readI = tempPc ;
                end
endmodule
```

6) mips_instruction_memory Instruction(readI , instruction);

Bir önceki kısımda açıkladığım modülden dönen readI degerini input olarak alır.Bu parametreyi index olarak düşünebiliriz.memory arrayinde tutulan instructionlardan hangisini bir sonraki olarak almak istediğimiz bu readI indexi ile belirlenir.

Mips_instruction_memory.v

```
module mips_instruction_memory( read_index , instruction);
        output reg[31:0] instruction;
        input [31:0] read_index;
        reg [31:0] memory [31:0];
        initial begin
                $readmemb(".\\instruction.mem", memory) ;
        end
        always @(read_index)
                begin
                        instruction = memory[read_index];
                end
endmodule
```

BENGİ YÖRÜKOĞLU
131044046