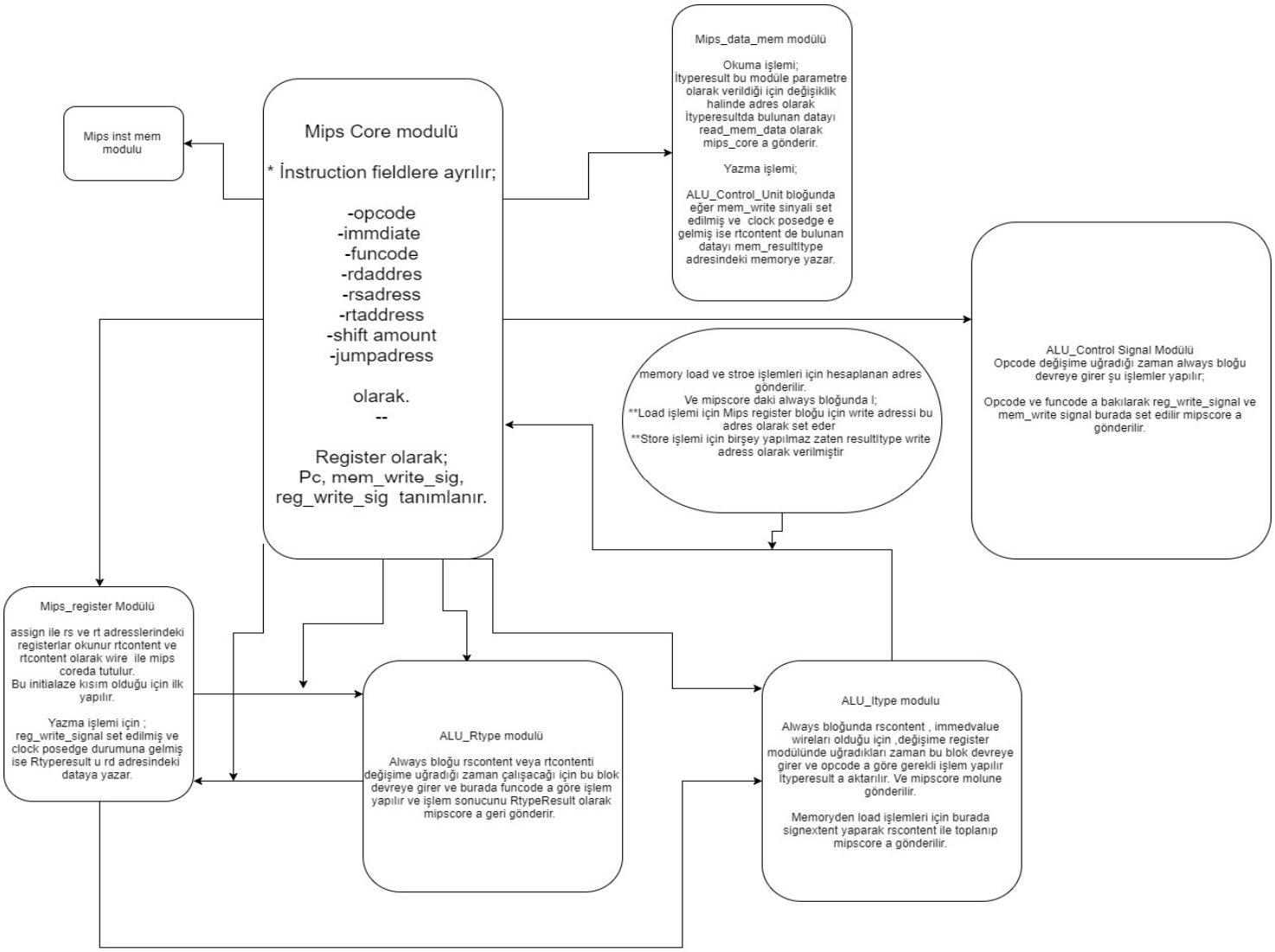
1. INTRODUCTION

1.1) Big Picture;



1.2) 1.2 Life cycle of 1 instruction

İlk olarak mem_instr_mem modülünden PC adresinde bulunan instr. Output olarak mips_core a verilir. Mips_core da ilk önce initialize işlemleri yapılır.

Modül içindeki rsadress,rtadress,rdadress,immvalue, sa, jumpadress gibi fieldler initialize edilir.

1) ilk olarak opcode değişime uğradığı için ALU_Control_Unit devreye girer ve verilen opcode ve funcode a göre mips_core da bulunan sig_mem_write ve signal_reg_write sinyalleri set edilir.
2) Hemen ardından Mips_core içinde, rs ve rt adressleri mips_register modulune parametre olarak verildiği için bu blok initialize edilmek üzere çalışır. Burada rsadress ve rtadress

lerindeki register contentleri ,rscontent ve rtcontent olarak output olarak mips_core modülüne gönderilir.

3) Mips_core içerisinde ALU_Rtype ve ALU_Itype modüllerine, rscontent ve rtcontent parametreleri verildiği için bu modüllerde bulunan always blokları ,değişiklik olması sebebiyle çalışır. Opcode a bakarak if case leri içinde hangisi opcode ile aynı ise o bloğun içerisine girer ve instr. için gerekli Rtype ve Itype olmalarına göre gerekli işlemler yapılır. ALU_Rtype için sonuç resultRtpe , ALU_Itype için sonuç resultItype ile mips_core içerisinde tutulur.
4) Mips_core da resultItype **mips_data_mem** modülüne parametre olarak gönderildiği ve değişikliğe uğradığı için modulun initialize kısmı çalışır ve assign ile datadaki resultItype adresinde bulunan datayı read_mem_data olarak mips_core da tutar.

   *Bu kısımdan sonra Mips_core içindeki always bloğunda yapılmış olan işlemler anlatılacaktır. Diğer modüllerin işlemleri tamamlanmıştır.

5) Mips_core daki submodullerin işlemlerini bittikten sonra mips_coreda bulunan always bloğu resultRtype ve resultItype ın değişimine bağlı olarak çalışacaktır.Burada eğer instr. **Rtype** ise mips_registerin writeadress =rdadress , writedata=resultRtype olarak set edilecektir. Ardından posedge clock zamanında mips_register içindeki always bloğu tekrar çalışacak ve sig_reg_write ALU_Control_unit modulunde set edildiği için rd adresine resultRtype datası register arrayına yazılacaktır .**I type** içinse eğer load veya store instr. değil iseler write adress olarak rtadress ,writedata olarak resultItype olarak seçilecektir. Ve clock posedge zamanında register bloğundaki always clockdan dolayı tekrar çalışıp sig_reg_write set edilmiş olması sebebiyle rtadressine resultItype ı yazacaktır.
6) 5.madde de belirtildiği gibi eğer instr. itype ve Load instructionu ise ; mips_core'un always bloğunda ,Önceden 4.maddede alınmış olan read_mem_data datasından **lbu** instructionu için sadece son 8 biti alınıp geri kalan kısımlara unsigned extend yapılarak ***writedata'a*** aktarılır. lhu inst. İçin read_mem_data'nın son 16 bit'i alınıp ,unsigned extend yapılarak writedata'ya aktarılır. En son lw için tüm read_mem_data writedata 'ya aktarılarak register modulunde posedge clock olduğu zaman register bloğuna yazılma işlemi yapılır.
7) 5.madde de belirtildiği gibi eğer inst.itype ve store instr. ise mip_core 'un always bloğu içinde bir işlem yapılmaz fakat posedge clock zamanı mips_data_mem içindeki always bloğunda store tipine göre (half,Word,byte) şeklinde dataya rtcontentinin belirtilmiş bölümü yazılır.

*Jump instructionu için mips_core içinde sadece PC 'ın set edildiği always bloğu içinde eğer opcode olarak eşit olursa Pc offset ile set edilecektir.

*Jal instruction için Control unitte sig_reg_write set edildiği için writeadress=11111 olarak belirlenip ,writedatası ise Pc+1 şeklinde 31.registera yazılır. En son Pc always bloğunda offset ile set edilir.

*Jr için Rtype da yapılan işlemler yapılır ardından , Pc rscontent ile set edilir.

NOT:

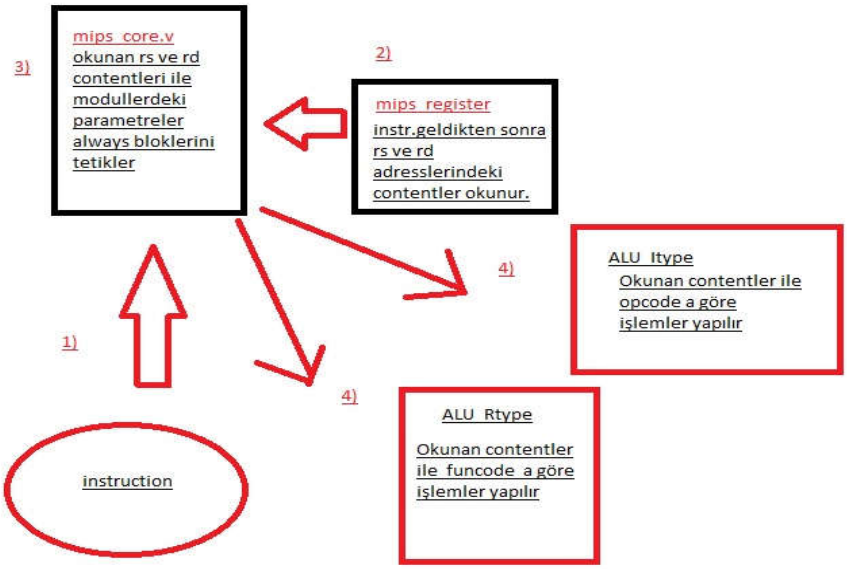
Sc instructionu Alp hocaya final günü sorulmuş ve gerekli olmadığını söylediği için implement edilmemiştir.

2. METHOD

Moduller içerisinde always blokları bulunmaktadır. Always blokları içindeki parametreye bağlı olarak bir modulun çıkış parametresi diğer modulun always bloğunu tetiklemektedir. Bu olay akışını kullanarak modülleri birbirine bağlama işlemini yaptım her bir parametrenin değişimi onunla alaklı diğer modülü çağırarak gerekli işlemleri yaptırmaktadır. Memory ve Register içine yazdırma işlemleri always bloğunda posedge clock içerisinde yapılmıştır. Clock negatif durumundayken, okuma ve hesaplama işlemleri ALU_rtype, ALU_Itype ve Mips_core tarafından yaptırılır. Ardından positif kısımda sadece yazma işlemi yapılmaktadır.

Birbirlerine bağlı modüller ;

2.1)For Step1;



2.2) For Step 2;

mips_core.v

```
reg [31:0] tempreg;
always @(resultRtype or resultItype)
begin
    if(opcode==6'b0)
    begin
        writeadd <=rdadd;
        ALUresult<=resultRtype;
    end
    else if(opcode==6'b001000 || opcode==6'b001001
    || opcode==6'b001100 || opcode==6'b001010 || opcod
    begin
        writeadd <=rtadd;
        ALUresult<=resultItype;

    end
    else if(opcode==6'b100011)//lw
    begin
        writeadd<=rtadd;
        ALUresult<=read_mem_data;

    end
    else if(opcode==6'b100100)//lbu
    begin
        writeadd<=rtadd;
        unsignedByte[31:8]={24{1'b0}};
        unsignedByte[7:0]=read_mem_data[7:0];
        ALUresult<=unsignedByte;
    end
    else if(opcode==6'b100101)//lhu
    begin
        writeadd<=rtadd;
        unsignedByte[31:16]={16{1'b0}};
```

```
ALU_Control_Unit control_unit(opcode,funcode,signal_reg_write,sig_mem_write);
mips_instr_mem instructionmem(instruction, PC);
mips_registers registers(rscontent,rtcontent,ALUresult,rsadd,rtadd,writeadd,signal_reg_write,clock);
ALU_Rtype Rtypecalculator(rscontent,rtcontent,sa,opcode,funcode,resultRtype,clock);
ALU_Itype Itypecalculator(rscontent,rtcontent,immvalue,opcode,resultItype,clock);
mips_data_mem memModule(read mem data, resultItype, rtcontent, sig_mem_write,clock,opcode);
```

ALU_Rtype.v

Bu blokda funcode'a göre gerekli aritmetik işlemler yapılır. Ardından mips_core da always bloğu tetiklenerek writeaddres ve writedata bilgileri set edilir.

ALU_Itype.v

Bu blokda opcode'a göre gerekli aritmetik işlemler yapılır. Ardından mips_core da always bloğu tetiklenerek writeaddres ve writedata bilgileri set edilir.

2.3) For Step 3;

mips_data_mem.v

```
assign read_data= data_mem[mem_address];
reg [31:0] tempreg;
always @( posedge clock ) begin

  if (sig_mem_write==1'b1)
  begin
     if(opcode==6'b101000)//sb
     begin
        tempreg =data_mem[$signed(mem_address)];
        tempreg[7:0]=write_data[7:0];
        data_mem[$signed(mem_address)] <=  tempreg;

     end
     else if (opcode==6'b101001) //sh
     begin
           tempreg[31:0]=data_mem[$signed(mem_address)][31:0];
           tempreg[15:0]=write_data[15:0];
           data_mem[mem_address] = tempreg;

     end
     else //sw
     begin
        data_mem[mem_address] = write_data[31:0];
     end
     $writememb("res_data.mem",data_mem);
  end

end
```

Tüm modüller aritmetik işlemlerini bitirir ve clock posedge durumuna geçer bu durumda sadece register ve dataya yazılma işlemi yapılır.

Posedge clock zamanında store type a göre rt content memory e yazılma işlemi yapılır.

3. RESULT

3.1 Testbench Results

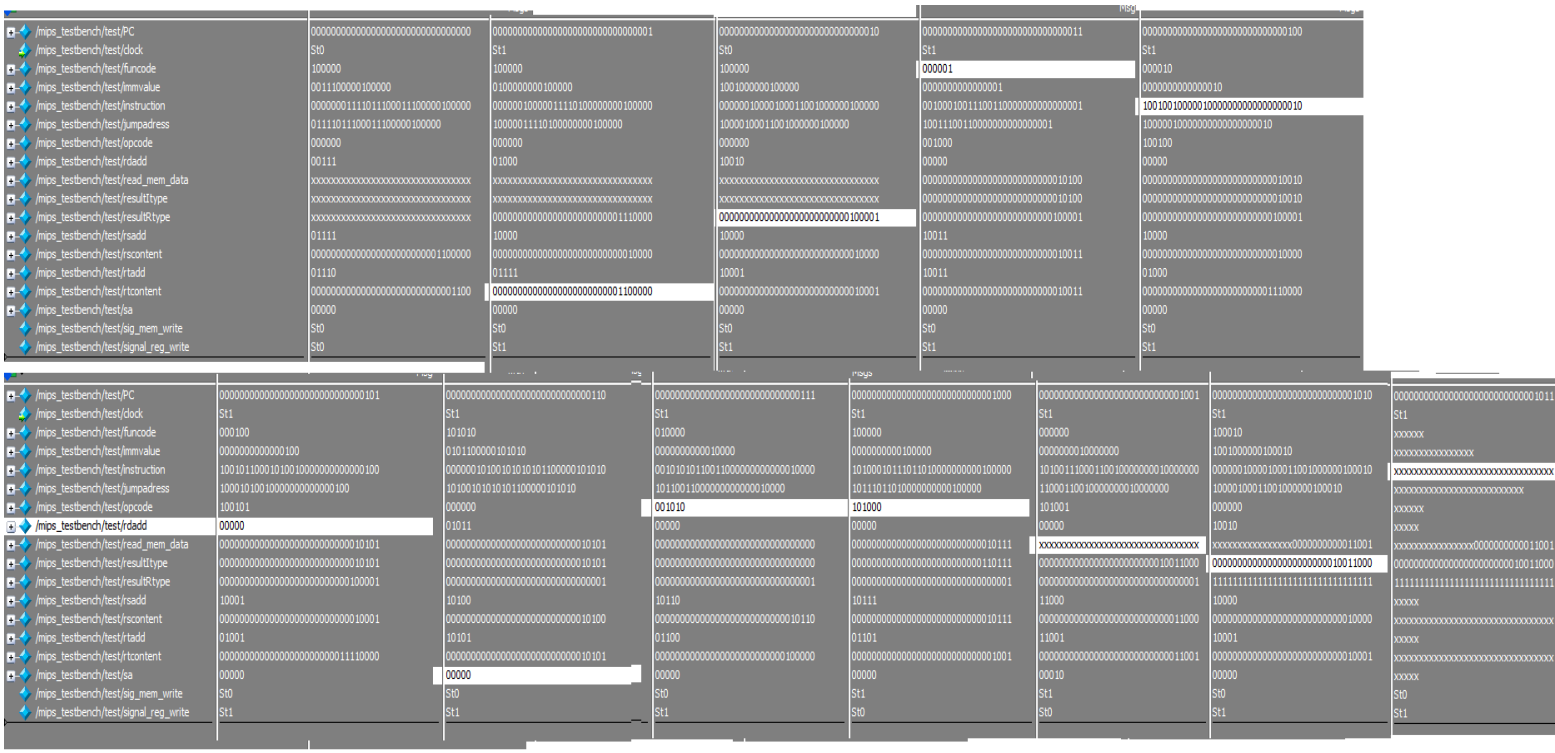
Test olarak vermiş olduğunuz 11 tane instruction örneği denenmiştir.

Sonuçlar testsonuclari klasörü altında bulunmaktadır. Ayrıntılı bakalirsiniz.

Monitor çıktısı;

```
# rscontent: 00000000000000000000000000010000 , rtcontent : 00000000000000000000000001100000 , opcode: 000000 ,
funcode :    32,  , resultRtype=00000000000000000000000001110000 ,  sig_write_reg=1

# rscontent: 00000000000000000000000000010000 , rtcontent : 00000000000000000000000000010001 , opcode: 000000 ,
funcode :    32,  , resultRtype=00000000000000000000000000100001 ,  sig_write_reg=1

# rscontent: 00000000000000000000000000010011 , rtcontent : 00000000000000000000000000010011 , opcode: 001000 ,
funcode :     1,  , resultRtype=00000000000000000000000000100001 ,  sig_write_reg=1

# rscontent: 00000000000000000000000000010000 , rtcontent : 00000000000000000000000001110000 , opcode: 100100 ,
signextend :0000000000000010,  , resultItype=00000000000000000000000000010010
,read_mem_data=00000000000000000000000000010010 ,  sig_write_mem=0 ,sig_write_reg=1

# rscontent: 00000000000000000000000000010001 , rtcontent : 00000000000000000000000011110000 , opcode: 100101 ,
signextend :0000000000000100,  , resultItype=00000000000000000000000000010101
,read_mem_data=00000000000000000000000000010101 ,  sig_write_mem=0 ,sig_write_reg=1
# rscontent: 00000000000000000000000000010100 , rtcontent : 00000000000000000000000000010101 , opcode: 000000 ,
signextend :0101100000101010,  , resultItype=00000000000000000000000000010101
,read_mem_data=00000000000000000000000000010101 ,  sig_write_mem=0 ,sig_write_reg=1

# rscontent: 00000000000000000000000000010110 , rtcontent : 00000000000000000000000000100000 , opcode: 001010 ,
funcode :    16,  , resultRtype=00000000000000000000000000000001 ,  sig_write_reg=1

# rscontent: 00000000000000000000000000010111 , rtcontent : 00000000000000000000000000001001 , opcode: 101000 ,
signextend :0000000000100000,  , resultItype=00000000000000000000000000110111
,read_mem_data=00000000000000000000000000010111 ,  sig_write_mem=1 ,sig_write_reg=0

# rscontent: 00000000000000000000000000011000 , rtcontent : 00000000000000000000000000011001 , opcode: 101001 ,
signextend :0000000010000000,  , resultItype=00000000000000000000000010011000
,read_mem_data=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx ,  sig_write_mem=1 ,sig_write_reg=0

# rscontent: 00000000000000000000000000010000 , rtcontent : 00000000000000000000000000010001 , opcode: 000000 ,
signextend :1001000000100010,  , resultItype=00000000000000000000000010011000
,read_mem_data=xxxxxxxxxxxxxxxx0000000000011001 ,  sig_write_mem=0 ,sig_write_reg=1

# rscontent: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx , rtcontent : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx , opcode: xxxxxx , funcode :     x,  ,
resultRtype=11111111111111111111111111111111 ,  sig_write_reg=1
```

Debug Çıktısı;



3.2 ANALYSIS

Tüm instructionlar doğru şekilde çalışmış olup , doğru sonuçlar vermiştir. Denenmeyen diğer instructionlar implement edilirken test edilmiştir. Ve bu instructionlara benzer olduğu için ayrıca test dosyası konulmamıştır.