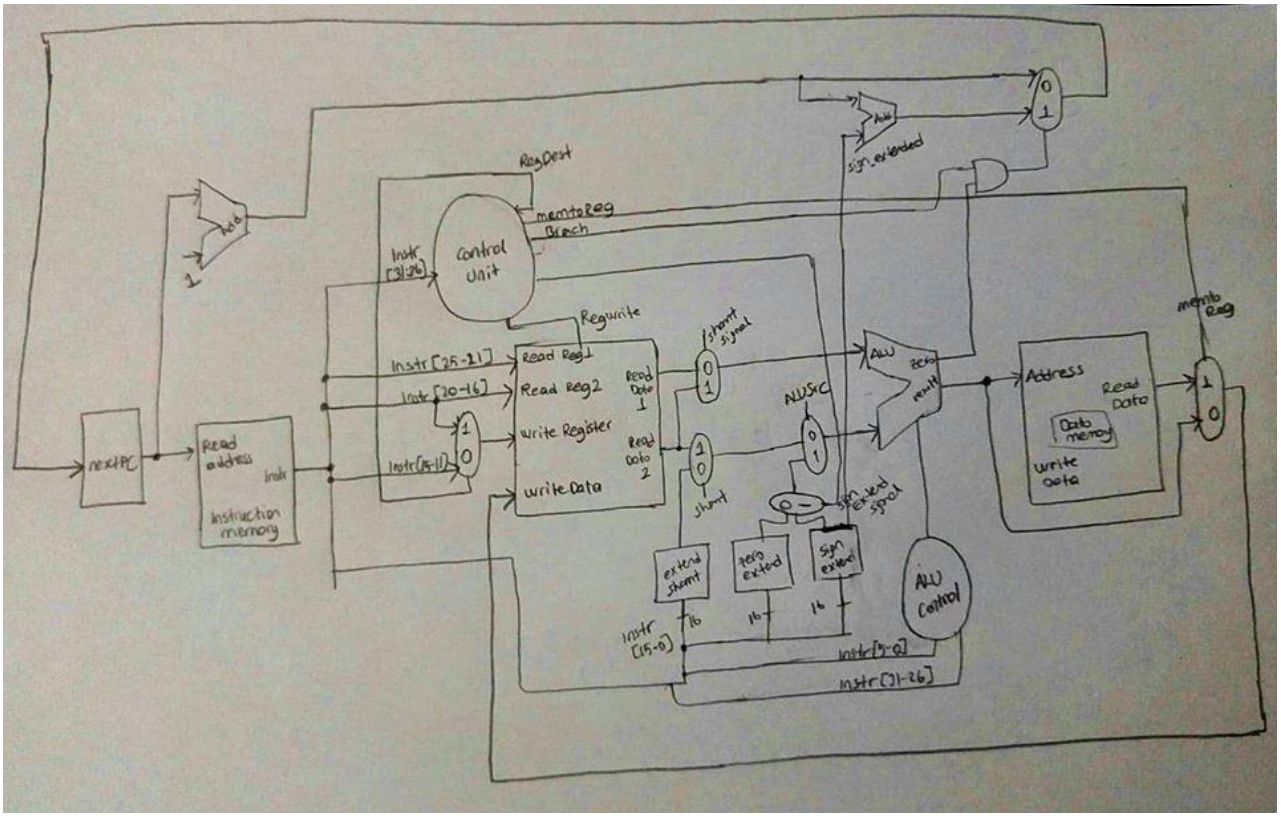
Gebze Teknik Üniversitesi
Bilgisayar Mühendisliği

CSE 331
FINAL PROJE RAPORU

Simge SARIÇAYIR
151044035

Dersin Asistanı :Fatma Nur Esirci

1. Single Cycle Datapath



Yukarıda assignment 4 için istenen instructionlar için bir önceki projedeki datapathe eklenen değişiklikler ile tasarımı yapılan datapath gözükmektedir.

Bir instructionın datapathteki işlem sırası:

- **mips32_single_cycle** modülü ilk olarak instruction memoryden instructionı alır.Instruction memeoryden instruction program counterin gösterdiği adres ile alınır.
- Data sonra control unite instructionun [31:26] bitleri yani opcode'u gönderilir ve burada sinyaller oluşturulur.
- Control unitte oluşan sinyallerden biri olan regDest sinyaline göre 5 bit multiplexer ile destination register seçilir.(rd veya rt)
- Alu control modülüne instructionun opcode ve function code bitleri gönderilir ve ALU için select bit oluşturulur.Ayrıca bu modülde shamt sinyali ve sltu için sinyal de üretilir.
- ALU'ya girecek datalar için 32 bit multiplexer ile seçim yapılır.

  Shamt sinyali 1 ise read data1 için rt seçilir.0 ise read data1 için rs seçilir.

  5 bit Shamt değeri 32 bite extend edilir.

  Shamt sinyali 1 ise read data2 için extend edilmiş shamt değeri seçilir. Shamt sinyali 0 ise rt seçilir.

- Instructionun [15:0] bitleri gönderilerek sign extend ve zero extend modüllerinden extend edilmiş halleri alınır.

- Sıgn extend ile zero extend resultu arasında multiplexer ile seçim yapılır sign extend sinyali 1 ise signExtend seçilir 0 ise zeroExtend seçilir.
- ALUSrc sinyaline göre alu read data2 için son seçim yapılır. ALUSrc 1 ise extend edilmiş sonuçlardan seçilen ALU'nun read data2 girişine verilir. ALUSrc 0 ise read data2 için rt'nin contenti gönderilir.
- ALU'ya girecek datalar seçildikten sonra ALU modülü çağırılır ve result elde edilir.
- ALU'dan gelen sonuç data memory'e adress olarak gönderilir.Data memory gerekli sinyal var ise okuma ya da yazma işlemini gerçekleştirir.
- ALU'dan gelen result sltu instructionu için concatane moduülüne gönderilip concatane edilmiş sonuç bir wire'da hazırda tutulur.
- 32 bit multiplexer kullanarak concatane edilmiş sonuç ile alu result arasında sltu sinyaline göre seçim yapılır.sltu sinyali 1 ise yani instruction sltu ise concatane edilmiş sonuç seçilir.Değil ise aludan gelen sonuç mips32_single_cycle modülünün outputu olan result'a seçilir.
- jump_address modülüne instructiounun [25:0] bitleri ile program counterın [31:28] bitleri gönderilir ve jump instructionu için adress hesaplanır.
- Beq instructionu için sinyal 1 ise ve alu zero biti 1 ise sign_Extended ile PC arasında mux ile seçim yapılırak sign_Extended seçilir.Sinyal 0 ise PC seçilir ve sonuç wireda tutulur.
- Beq için muxtan gelen wiredaki sonuç ile jump adress arasında jump sinyaline göre mux ile seçim yapılır.jump sinyali 1ise jump adress seçilir.0 ise bir önceki muxtan gelen sonuç seçilir ve bu değer nextPC değeri olur.

2. Modüller

2.1. instr_mem(instruction ,pc);

Instruction memory modülüdür.Input olarak pc alır ve output olarak program counterın gösterdiği adresteki 32 bit instructionı verir.

```
module instr_mem(instruction ,pc);
input [31:0] pc;
output reg [31:0] instruction;


reg [31:0] instr_mem [255:0];

//always takes the instruction with program counter
   always @(*) begin
      instruction = instr_mem[pc];
   end

endmodule
```

2.2. mips_registers

Dosyadan okunmuş olan registerlar arasından input olarak gelen read data adreslerinden 32 bit content output olarak verilir. signal_reg_write 1 ise ve write register zero registerı değilse

input olarak gelen write data clock değişiminde yine input olarak gelen write_reg adresine yazılır.

```
    output [31:0] read_data_1, read_data_2;
    input [31:0] write_data;
    input [4:0] read_reg_1, read_reg_2, write_reg;
    input signal_reg_write, clk;

    reg [31:0] registers [31:0];


    assign read_data_1 = registers[read_reg_1];
    assign read_data_2 = registers[read_reg_2];

    always@(posedge clk)
    if( signal_reg_write && write_reg!=5'b0) begin
        registers[write_reg] = write_data;
       end

endmodule
```

2.3. data_mem

İnput olarak yazılacak veya okunacak adres, clock ve memRead ve memWrite sinyalleri alır.

Output verme durumu yalnızca load instructionlarında olacağından yani okunan datanın dışarı verileceği zamanda olur.Bu yüzden bir tane output vardır.

```
output reg [31:0] read_data;
input [31:0] mem_address,write_data;
input memRead_signal,memWrite_signal,clock;

reg [31:0] data_mem  [255:0];

// mem read sinyali 1 ise verilen adresteki content read dataya atandı.
always @(*) begin
   if (memRead_signal) begin
       read_data[31:0] = data_mem[mem_address];
   end
end

// mem write sinyali 1 ise data memorydeki verilen adrese write data atandı.
always @(posedge clock) begin
   if (memWrite_signal) begin
      data_mem[mem_address] = write_data[31:0];
   end
end

endmodule
```

2.4 nextPC

İnput olarak 32 bit PC adresi alır. Bunun yanında jump ve branch için sinyal alır.

Jump ise input olarak gelen adresi nextPc adresi yapar.(Jump ise curentpc mips32_single_cycle modülünden jump yapılmak istenen adres olarak gönderilir.)

Branch ise nextPC adresi branch adresidir.Bu yüzden currentpc ile toplanarak nextPC adresine atanır branch.Program counter instructionun atlamak istediği yeri gösterir.

```
module nextPC(nextPC,clock,jump_signal,branch_signal,CurrentPC);
  input clock,branch_signal,jump_signal;
  input [31:0] CurrentPC;
  output reg [31:0] nextPC;

  always @ (posedge clock) begin
      if(jump_signal==1) begin//jump ise gelen adresi bir sonraki adres yap.
         nextPC = CurrentPC;
      end
      // beq için kontrol
      else if(branch_signal == 1) begin
         nextPC = nextPC + CurrentPC;
      end
      else begin// program counterı bir arttır sıradaki instr al.
         nextPC = nextPC+1;
      end
  end
endmodule
```

2.5 jump_address

Jump instructionu için 32 bit adress outputu oluşturulan modüldür.İnput olarak instructinoun [25:0] bitlerini alır yani jump instructionunun atlamak istediği adressi gösteren bitleri almış olur.Ek olarak program counterın son 4 bitini alır.

Sonucun 32 bit olması gerektiği için bu 26 bit output olacak jump adresinin ilk 26 bitine konulur.27 ve 28. Bitlerine 0 konulur.Geriye kalan 4 bit ise program counterın son 4 bitidir.

2.6 control_unit

Input olarak instructionun [31:25] yani opcode bitlerini alır ve aşağıdaki sinyalleri output olarak verir.

- memRead_signal : 1 olduğundan data memoryden data okumaya izin verem sinyalidir.Yalnızca lw için 1 verir.
- memWrite_signal : 1 olduğundan data memory'e yazmaya izin veren sinyalidir.Yalnızca sw sinyali için 1 verir.
- memtoReg : yalnızca lw için 1 sinyali üretir.Data memoryden registera yazılacağını ifade eder.
- regWrite_signal : mips registera yazma sinyalidir.R-type instructionlar için her zaman 1dir. addiu , lw, andi ve ori için de 1 sonucunu verir.Diğer tüm instructionlar için 0 sonucunu verir.

- signExtend_signal : instructionun [15:0] bitlerinin sign extend edilmiş sonucu aluya gönderileceğini ifade eder. lw,sw,addiu instructionlarından biriyse signExtend_signal 1.Diğerleri için 0'dır.
- zeroExtend_signal : instructionun [15:0] bitlerinin zero extend edilmiş sonucu aluya gönderileceğini ifade eder. andi, ori instructionlarından biriyse zeroExtend_signal 1
- jump_signal : yalnızca jump instructionu için 1 verir.
- beq_signal : yalnızca beq instructionu için 1 verir.
- regDest : Destination registeri seçmek için üretilmiştir.addiu , lw, andi,ori instructionlarından biriyse regDest 1 yani rt'dir.Diğer tüm instructionlar için 0 yani rd'dir.
- ALUSrc : ALUSrc 1 ise Alu read data2 için extend edilmiş değeri seçer, 0 ise rt değerini seçer. addiu , lw ,sw, andiu veya ori ise ALUSrc 1 diğerleri için ALUSrc 0'dır.

2.7 alu_control

Input olarak function code ve opcode alır.R type instructionlar için önceki tasarımlarda olduğu gibi functione code'a göre ALU için select bit oluşturur.Oluşturulan select bit wireda tutulur.

I type ve J type instructionlar için opcode'a göre(000000 olmadığı için) ALU için select bit oluşturur. Oluşturulan select bit wireda tutulur.

En son 3 bit mux kullanılarak bu iki select bit arasından seçim yapılır ve output seçilen 3 bit select olur.R type ise yani opcodeların hepsi 0 ise function code'a göre yapılan select bit seçilir diğer türlü opcode'a göre yapılan seçilir.

Ek olarak bu modül içerisinde function code'a bakılarak shamt ve sltu sinyali üretilir.

2.8 zero_extend

Instructionun [15:0] immediate parçasını input olarak alır ve 0 ile concatane ederek 32 bit extend edilmiş halini output olarak verir.

2.9 sign_extend

Instructionun [15:0] immediate parçasını input olarak alır ve most significant biti ile concatane ederek 32 bit extend edilmiş halini output olarak verir.

2.10 alu32

Alu modülü bir önceki projedeki tasarım ile aynıdır bir değişiklik yapılmamıştır.Alu modülü input olarak 32 bit read data1 ve read data 2 alır.Bu iki inputu andop ,orop, carry_ripple_adder_subtractor , xorop,left_shifter ve right_shifter(Shifte select bit seçilecek shift bit sıfır ise 0 bir ise 1 ile shift etmek için mux kullanır.sra srl kontrolü yapılmış olur),nor modüllerine gönderip hepsinden sonuçlarını wire ile alır.Daha sonra diğer bir input olan

ALUOP 3 bit select ile 8x1 mux kullanarak doğru operationun sonucunu output result olarak verir.Ayrıca overflow ve zero için de iki tane output verir.

3.Testbench

```
`timescale 10ps/1ps
module testbench();
   wire [31:0] R;
   wire result;

   reg clk2;
   reg [7:0] index;
   mips32_single_cycle i0(.clock(clk2),.result(R));

   always
   begin
    #15 clk2=~clk2;
   end

   initial
   begin
      clk2=0;
      $readmemb("instruction.mem", i0.instructions.instr_mem);
      $readmemb("registers.mem", i0.mips.registers);
      $readmemb("data.mem", i0.data_memory.data_mem);
      i0.pr.nextPC= 32'b0;
      index = 0;
   end
```

İnstructionlar simulation/modelsim/instruction.mem dosyasınan okunur.

Registerlar simulation/modelsim/registers.mem dosyasınan okunur.

Data memory simulation/modelsim/data.mem dosyasınan okunur.

3.1 instruction.mem

```
00000010000100011000100000100000
00000010100101011001100000100001
00000001001010100110100000100111
00000001000010010101000000100101
00000010110101111011110000010101 1
00000000000101001000000010000000
00000000000010000100000011000010
00000010000100011100000000100010
00000010110010001100100000100011
00000010101011100100000000100100
00100100001001010000000000000010
10001100001001000000000000000001
10101100011010000000000000000011
00110000110001100000000000001001
00010000000000000000000000000001
00110101000001110011001100001100
00001000000000000000000000010001
00000010000100011000100000100000
```

1. add $17,$16,$17
2. addu $19,$20,$21
3. nor $13,$9,$10
4. or $10,$8,$9
5. sltu $15,$22,$23
6. sll $16,$20,2
7. srl $16,$16,3
8. sub $24,$16,$17
9. subu $25,$22,$8
10. and $8,$21,$14
11. addiu $5,$21,2
12. lw $4,1($1)
13. sw $3,3($8)
14. andi $6,$6,9
15. beq $0,$0,16
16. ori $7,$8,0011001100001100
17. jump 18
18. add $17,$16,$17

3.2 Simulation Results

3.2.1 registers.mem and regLast.mem

registers.mem

```
1  00000000000000000000000000000000
2  00000000000000000000000000000001
3  00000000000000000000000000000010
4  00000000000000000000000000000011
5  00000000000000000000000000000100
6  00000000000000000000000000000101
7  00000000000000000000000000000110
8  00000000000000000000000000000111
9  00000000000000000000000000001111
10 00000000000000000000000011110000
11 00000000000000000000111100000000
12 00000000000000000001000000000000
13 00000000000000000000000100000000
14 00000000000000000000000000001001
15 00000000000000000000000000001100
16 00000000000000000000000001100000
17 00000000000000000000000000010000
18 00000000000000000000000000010001
19 00000000000000000000000000010010
20 00000000000000000000000000010011
21 00000000000000000000000000010100
22 00000000000000000000000000010101
23 00000000000000000000000000010110
24 00000000000000000000000000010111
25 00000000000000000000000000011000
26 00000000000000000000000000011001
27 00000000000000000000000000011010
28 00000000000000000000000000011011
29 00000000000000000000000000011100
30 00000000000000000000000000011101
31 00000000000000000000000000011110
32 00000000000000000000000000011111
```

regLast.mem

```
1  // memory data file (do not edit the
2  // instance=/testbench/i0/mips/regist
3  // format=bin addressradix=h dataradi
4  00000000000000000000000000000000
5  00000000000000000000000000000001
6  00000000000000000000000000000010
7  00000000000000000000000000000011
8  00000000000000000000000000000010
9  00000000000000000000000000000011
10 00000000000000000000000000000000
11 00000000000000000011001100001100
12 00000000000000000000000000000100
13 00000000000000000000000011110000
14 00000000000000000000000011111111
15 00000000000000000001000000000000
16 00000000000000000000000000100000
17 11111111111111111111000000001111
18 00000000000000000000000000001100
19 00000000000000000000000000000001
20 00000000000000000000000000001010
21 00000000000000000000000000101011
22 00000000000000000000000000010010
23 00000000000000000000000000101001
24 00000000000000000000000000010100
25 00000000000000000000000000010101
26 00000000000000000000000000010110
27 00000000000000000000000000010111
28 11111111111111111111111111101001
29 00000000000000000000000000000111
30 00000000000000000000000000011010
31 00000000000000000000000000011011
32 00000000000000000000000000011100
33 00000000000000000000000000011101
34 00000000000000000000000000011110
```

Instruction.memdeki instructionlara göre kırmızı renk ile gösterilen registerlara son yazma işlemi yapılır.

- İndex = 1
  add $17,$16,$17
  result = 00000000000000000000000000100001 17. Registera yazılır.
- İndex = 2
  addu $19,$20,$21
  result = 00000000000000000000000000101001 19.registera yazılır.
- İndex = 3
  nor $13,$9,$10
  result = 11111111111111111111000000001111 13. Registera yazılır.

- İndex = 4
  or $10,$8,$9
  result = 00000000000000000000000011111111 10.registera yazılır.
- İndex =5
  sltu $15,$22,$23
  result = 00000000000000000000000000000001 15. registera yazılır.
- İndex = 6
  sll $16,$20,2
  result = 00000000000000000000000001010000 16.registera yazılır
- İndex = 7
  srl $16,$16,3
  result = 00000000000000000000000000001010 16. Registera yazılır.
- İndex = 8
  sub $24,$16,$17
  result = 11111111111111111111111111101001 24. Registera yazılır.
- İndex = 9
  subu $25,$22,$8
  result = 00000000000000000000000000000111 25. Regisera yazılır.
- İndex =10
  and $8,$21,$14
  result = 00000000000000000000000000000100 8.registera yazılır.
- İndex = 11
  addiu $5,$21,2
  result = 00000000000000000000000000000011 5. Registera yazılır.
- İndex = 12
  lw $4,1($1)
  result = 00000000000000000000000000000010 4. Registera yazılır.
- İndex = 13
  sw $3,3($8)
  result = 00000000000000000000000000000110 data memorye yazar.
- İndex =14
  andi $6,$6,9
  result = 00000000000000000000000000000000 6. Registera yazar.
- İndex = 15
  beq $0,$0,15 16. indexteki instructiona atlar.
- İndex = 16
  ori $7,$8,0011001100001100
  result = 00000000000000000011001100001100 7. Registera yazar.
- İndex = 17
  jump 17 18 indexteki. İnstructiona atlar
- İndex = 18
  add $17,$16,$17
  result = 00000000000000000000000000101011 17.registera yazar.

3.2.2 data.mem and dataLast.mem

simulation/modelsim/data.mem

```
1  00000000000000000000000000000000
2  00000000000000000000000000000001
3  00000000000000000000000000000010
4  00000000000000000000000000000011
5  00000000000000000000000000000100
6  00000000000000000000000000000101
7  00000000000000000000000000000110
8  00000000000000000000000000000111
9  00000000000000000000000000001000
10 00000000000000000000000000001001
11 00000000000000000000000000001010
12 00000000000000000000000000001011
13 00000000000000000000000000001100
14 00000000000000000000000000001101
15 00000000000000000000000000001110
16 00000000000000000000000000001111
17 00000000000000000000000000010000
18 00000000000000000000000000010001
19 00000000000000000000000000010010
20 00000000000000000000000000010011
21 00000000000000000000000000010100
22 00000000000000000000000000010101
23 00000000000000000000000000010110
24 00000000000000000000000000010111
25 00000000000000000000000000011000
26 00000000000000000000000000011001
27 00000000000000000000000000011010
28 00000000000000000000000000011011
29 00000000000000000000000000011100
30 00000000000000000000000000011101
31 00000000000000000000000000011110
32 00000000000000000000000000011111
33 00000000000000000000000000000000
34 00000000000000000000000000000001
```

simulation/modelsim/dataLast.mem

```
1  // memory data file (do not edit the
2  // instance=/testbench/i0/data_memory
3  // format=bin addressradix=h dataradi
4  00000000000000000000000000000000
5  00000000000000000000000000000001
6  00000000000000000000000000000010
7  00000000000000000000000000000011
8  00000000000000000000000000000100
9  00000000000000000000000000000101
10 00000000000000000000000000000100
11 00000000000000000000000000000111
12 00000000000000000000000000001000
13 00000000000000000000000000001001
14 00000000000000000000000000001010
15 00000000000000000000000000001011
16 00000000000000000000000000001100
17 00000000000000000000000000001101
18 00000000000000000000000000001110
19 00000000000000000000000000001111
20 00000000000000000000000000010000
21 00000000000000000000000000010001
22 00000000000000000000000000010010
23 00000000000000000000000000010011
24 00000000000000000000000000010100
25 00000000000000000000000000010101
26 00000000000000000000000000010110
27 00000000000000000000000000010111
28 00000000000000000000000000011000
29 00000000000000000000000000011001
30 00000000000000000000000000011010
31 00000000000000000000000000011011
32 00000000000000000000000000011100
33 00000000000000000000000000011101
34 00000000000000000000000000011110
```

Yalnızca 13. İndexteki sw instructionu data memory'e yazar.

- İndex = 13
  sw $3,3($8)
  result = 00000000000000000000000000000110 data memorye yazar.
  Filedaki 7. indexe yazar.

```
VSIM 15> step -current
#
# opcode = 000000, rs = 10000, rt = 10001, rd = 10001, shamt = 00000, funct = 100000 ,shamt_signal== 0,sltu=0 ,index==         1
# ReadData1 = 00000000000000000000000000010000
# ReadData2 =00000000000000000000000000010001
# result = 00000000000000000000000000100001
#
# opcode = 000000, rs = 10100, rt = 10101, rd = 10011, shamt = 00000, funct = 100001 ,shamt_signal== 0,sltu=0 ,index==         2
# ReadData1 = 00000000000000000000000000010100
# ReadData2 =00000000000000000000000000010101
# result = 00000000000000000000000000101001
#
# opcode = 000000, rs = 01001, rt = 01010, rd = 01101, shamt = 00000, funct = 100111 ,shamt_signal== 0,sltu=0 ,index==         3
# ReadData1 = 00000000000000000000000011110000
# ReadData2 =00000000000000000000111100000000
# result = 11111111111111111111000000001111
#
# opcode = 000000, rs = 01000, rt = 01001, rd = 01010, shamt = 00000, funct = 100101 ,shamt_signal== 0,sltu=0 ,index==         4
# ReadData1 = 00000000000000000000000000001111
# ReadData2 =00000000000000000000000011110000
# result = 00000000000000000000000011111111
#
# opcode = 000000, rs = 10110, rt = 10111, rd = 01111, shamt = 00000, funct = 101011 ,shamt_signal== 0,sltu=1 ,index==         5
# ReadData1 = 00000000000000000000000000010110
# ReadData2 =00000000000000000000000000010111
# result = 00000000000000000000000000000001
#
# opcode = 000000, rs = 00000, rt = 10100, rd = 10000, shamt = 00010, funct = 000000 ,shamt_signal== 1,sltu=0 ,index==         6
# ReadData1 = 00000000000000000000000000010100
# ReadData2 =00000000000000000000000000000010
# result = 00000000000000000000000001010000
#
# opcode = 000000, rs = 00000, rt = 10000, rd = 10000, shamt = 00011, funct = 000010 ,shamt_signal== 1,sltu=0 ,index==         7
# ReadData1 = 00000000000000000000000001010000
# ReadData2 =00000000000000000000000000000011
# result = 00000000000000000000000000001010
#
# opcode = 000000, rs = 10000, rt = 10001, rd = 11000, shamt = 00000, funct = 100010 ,shamt_signal== 0,sltu=0 ,index==         8
# ReadData1 = 00000000000000000000000000001010
# ReadData2 =00000000000000000000000000100001
# result = 11111111111111111111111111101001
#
```

```
#
# opcode = 000000, rs = 10110, rt = 01000, rd = 11001, shamt = 00000, funct = 100011 ,shamt_signal== 0,sltu=0 ,index==         9
# ReadData1 = 00000000000000000000000000010110
# ReadData2 =00000000000000000000000000001111
# result = 00000000000000000000000000000111
#
# opcode = 000000, rs = 10101, rt = 01110, rd = 01000, shamt = 00000, funct = 100100 ,shamt_signal== 0,sltu=0 ,index==        10
# ReadData1 = 00000000000000000000000000010101
# ReadData2 =00000000000000000000000000001100
# result = 00000000000000000000000000000100
#
# opcode = 001001, rs = 00001, rt = 00101,immediate = 0000000000000010 ,index =         11
# ReadData1 = 00000000000000000000000000000001
# ReadData2 =00000000000000000000000000000010
# result = 00000000000000000000000000000011
#
# opcode = 100011, rs = 00001, rt = 00100,immediate = 0000000000000001 ,index =         12
# ReadData1 = 00000000000000000000000000000001
# ReadData2 =00000000000000000000000000000001
# result = 00000000000000000000000000000010
#
# opcode = 101011, rs = 00011, rt = 01000,immediate = 0000000000000011 ,index =         13
# ReadData1 = 00000000000000000000000000000011
# ReadData2 =00000000000000000000000000000011
# result = 00000000000000000000000000000110
#
# opcode = 001100, rs = 00110, rt = 00110,immediate = 0000000000001001 ,index =         14
# ReadData1 = 00000000000000000000000000000110
# ReadData2 =00000000000000000000000000001001
# result = 00000000000000000000000000000000
#
# opcode = 000100, rs = 00000, rt = 00000,immediate = 0000000000000001 ,index =         15
# ReadData1 = 00000000000000000000000000000000
# ReadData2 =00000000000000000000000000000000
# result = 00000000000000000000000000000000
#
# opcode = 001101, rs = 01000, rt = 00111,immediate = 0011001100001100 ,index =         16
# ReadData1 = 00000000000000000000000000000100
# ReadData2 =00000000000000000011001100001100
# result = 00000000000000000011001100001100
#
# opcode = 000010  ,address =00000000000000000000010001  ,index =         17
# jumpAddress = 00000000000000000000000000010001
#
# opcode = 000000, rs = 10000, rt = 10001, rd = 10001, shamt = 00000, funct = 100000 ,shamt_signal== 0,sltu=0 ,index==        18
# ReadData1 = 00000000000000000000000000001010
# ReadData2 =00000000000000000000000000100001
# result = 00000000000000000000000000101011
#          18 tests completed.
#
# ** Note: $finish    : C:/Users/Lenovo/Desktop/project4_restored/testbench.v(50)
#    Time: 5250 ps  Iteration: 1  Instance: /testbench
# 1
```