NEVRA GÜRSES
161044071

COMPUTER ORGANIZATION PROJECT - 4

MODULES THAT I USED IN PREVIOUS PROJECT :

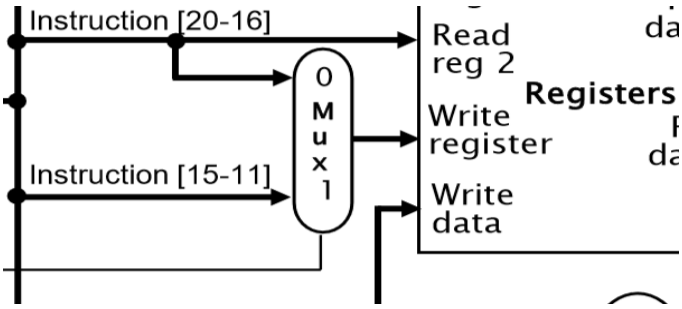
- mips_registers (read_data_1, read_data_2, write_data, read_reg_1, read_reg_2, write_reg, signal_reg_write, clk);
- mips_data_memory(read_data,address,data_in,opcode,clk,signal_mem_write,signal_mem_read);
- read_instruction(instruction, program_counter);
- loadOperations(load,readFromMem,lbSignal,lhSignal);
- luiControl(out,immediate,luiSignal,data);
- extended(immediate,extendedImmediate,signal);
- mux2_to_1_32Bit (muxOut ,I0,I1,signal);
- Pc(program_counter,clk,pcIn);
- Alu_32_Bit (carryoutputAlu,outputAluputAlu,Z,a,b,carryIn,AluOp);
- Alu_1Bit(carryOut,outputAlu,a,b,carryIn,less,AluOp);
- MSB_1BitAlu(carryOut,outputAlu,V,set,a,b,carryIn,less,AluOp);
- mux (muxOut ,I0,I1,I2,I3,s0,s1);
- xorGate (result,a ,b);

I used this modules in currently Project and I don't change them.

Now,I will explain my new modules and changed modules that I used in currently Project:

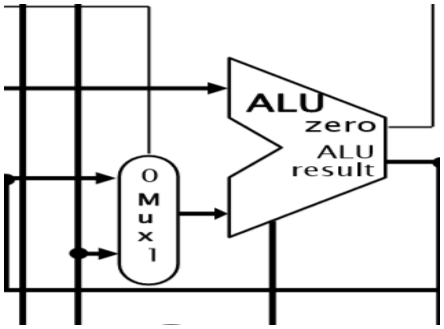
selectDestination(outputReg,rt,rd,RegDst): This module for selecting destination register.If instruction is I type, destination register is rt,if instruction is R type destination register is rd.

This module below mux part of datapath.



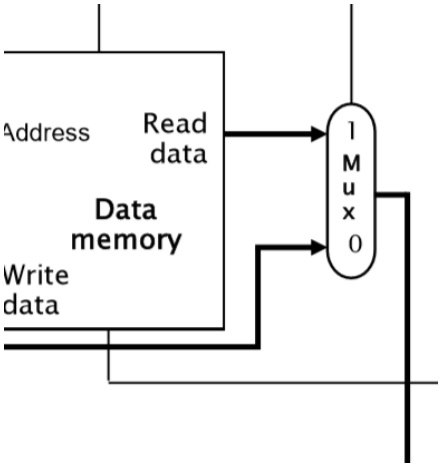
selectAluInputReg (outSelect,rt,extended,ALUSrc): This module for selecting one input of Alu that will process with rs content.If extended operation will done with rs content according to ALUSrc signal,this module selects extended content as input of Alu.If rt and rs operation will done in Alu, this module select rt register contents as input.

This module below mux part of datapath.



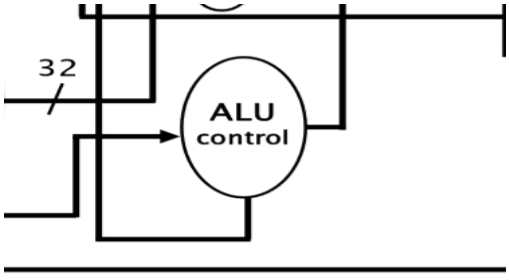
selectWriteReg (writeReg,outputAlu,memoryOutput,MemtoReg): This module selects writed data.If instruction is load instructions,memory content is written register.If instruction is R-type instruction, Alu result is written in register.

This module below mux part of datapath.



AluControl (outputAluControl,funct,AluOp): This module is Alu Control Unit.It selects true AluCtr according to instruction type.

This module below part of datapath.

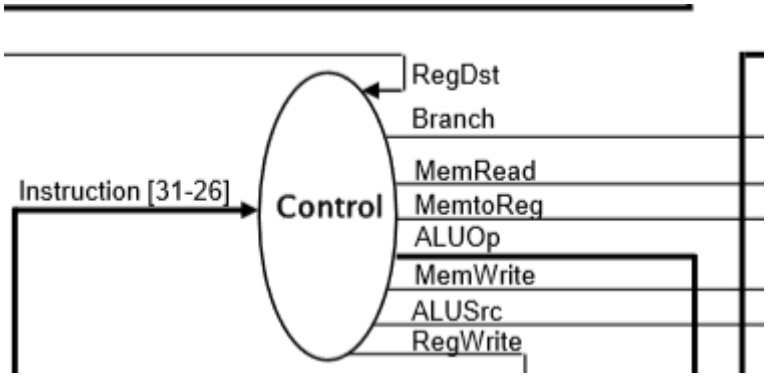


controlJump(jumpOut,fourBitPC,address): This module is apply jump operation.If jump signal is active,Pc is loaded with jump address.It is created with concataneting as

00-PC[31:28]-address.

controlUnit(MemtoReg,RegWrite,MemRead,MemWrite,extended,luiControl,load,opcode,branch,j,jal,jr,AluOp,AluSrc,regDest): This module is writed in previous Project.But for this Project,I modify it.New signals are added. These new signals are AluOp,AluSrc,j,jal,jr,regDest,branch.

This module below part of datapath.

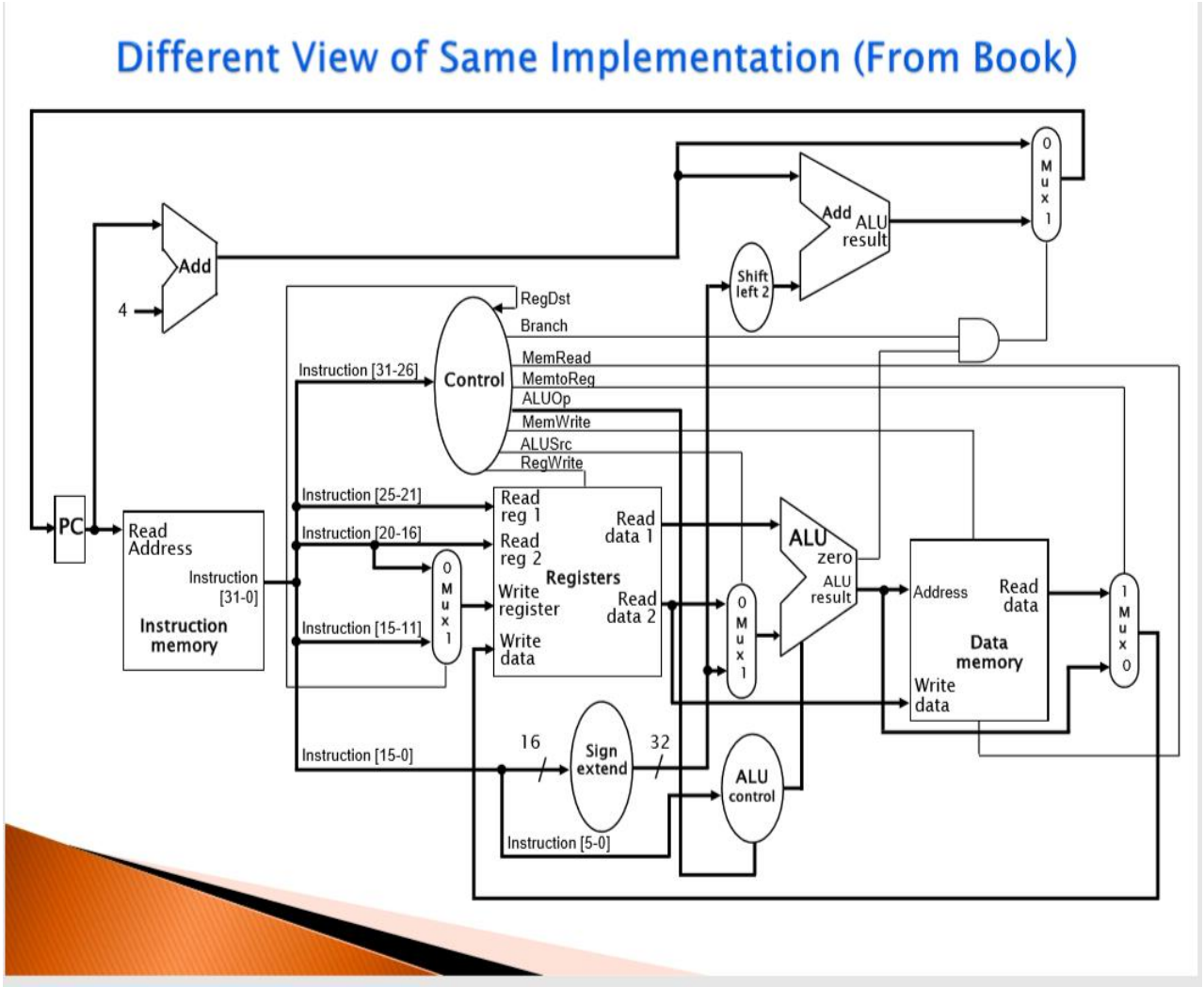


TestBench Of Control Unit:

```
sim:/controlUnit_testbench/regDest
VSIM 985> step -current
# opcode =100000 , memtoReg =1 , regWrite =1 , memRead =1 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0
# opcode =100100 , memtoReg =1 , regWrite =1 , memRead =1 , memWrite =0 , extended =1 , luiControl =0 , load =00, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0
# opcode =100001 , memtoReg =1 , regWrite =1 , memRead =1 , memWrite =0 , extended =0 , luiControl =0 , load =10, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0
# opcode =100101 , memtoReg =1 , regWrite =1 , memRead =1 , memWrite =0 , extended =1 , luiControl =0 , load =10, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0
# opcode =001111 , memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =1 , load =00, AluOp=00, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=0
# opcode =100011 , memtoReg =1 , regWrite =1 , memRead =1 , memWrite =0 , extended =1 , luiControl =0 , load =01, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0
# opcode =101000 , memtoReg =0 , regWrite =0 , memRead =0 , memWrite =1 , extended =1 , luiControl =0 , load =00, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0
# opcode =101001 , memtoReg =0 , regWrite =0 , memRead =0 , memWrite =1 , extended =1 , luiControl =0 , load =00, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0
# opcode =101011 , memtoReg =0 , regWrite =0 , memRead =0 , memWrite =1 , extended =1 , luiControl =0 , load =00, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0
# opcode =000000 , memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1
# opcode =000010 , memtoReg =0 , regWrite =0 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=00, AluSrc=0, branch=0, j=1, jal=0, jr=0, regDest=0
# opcode =000011 , memtoReg =0 , regWrite =0 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=00, AluSrc=0, branch=0, j=0, jal=1, jr=0, regDest=0
# opcode =001000 , memtoReg =0 , regWrite =0 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=00, AluSrc=0, branch=0, j=0, jal=0, jr=1, regDest=0
# opcode =000100 , memtoReg =0 , regWrite =0 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=01, AluSrc=0, branch=1, j=0, jal=0, jr=0, regDest=0

VSIM 986>
```

The big Picture is:



TESTING OF INSTRUCTIONS:

Testing “add” Operation:

Instruction memory is:

```
1   00000000000000010001000000100000
```

Opcode=000000 , rs= 00000 , rt=00001, shampt=00000, funct=100000

Before operation register memory is:

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips32_testbench/tb/register/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000100101
00000000000000000000000001011101
00000000000000000000000000000000
00000000000000001010010101010101
10100101010101010000000000000000
10000000000111111111000000001100
```

Testbench:

```
PC:  0, instruction: 00000000000000010001000000100000
opcode: 000000, rs: 00000, rt: 00001 , rd=00010,dest=00010
 memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 0
Register: read_data_1: 00000000000000000000000000100101, read_data_2: 00000000000000000000000001011101, write_data: 00000000000000000000000010000010
```

After add operation register memory is:

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips32_testbench/tb/register/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000100101
00000000000000000000000001011101
00000000000000000000000010000010
00000000000000001010010101010101
10100101010101010000000000000000
10000000000111111111000000001100
```

After Add operation, rs (00000) content and rt (00001) content is added.And it is writed in rd(00010).

Testing “and” Operation:

Intruction memory is:

Opcode=000000, rs= 00010, rt= 00011, rd= 00100 shampt=00000 funct=100100.

```
00000000010000110010000000100100
```

Before operation register memory is:

```
00000000000000000000000000100101
00000000000000000000000001011101
00000000000000000000000010000010
00000000000000001010010111010111
00000000000000000000000000000000
10000000000111111111000000001100
00000000000000000000000000000110
```

Testbench:

```
PC:  1, instruction: 00000000100001100100000000100100
opcode: 000000, rs: 00010, rt: 00011 , rd=00100,dest=00100
 memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
Register: read_data_1: 00000000000000000000000010000010, read_data_2: 00000000000000001010010111010111, write_data: 00000000000000000000000010000010
```

After operation register memory is:

```
00000000000000000000000000100101
00000000000000000000000001011101
00000000000000000000000010000010
00000000000000001010010111010111
00000000000000000000000010000010
10000000000111111111000000001100
```

After and operation register content of rs (00010) and register content of rt(00011) is anded and result is writed in rd(00100).

Testing “or” Operation:

Intruction memory is:

Opcode=000000, rs= 00100, rt= 00101, rd= 00110 shampt=00000 funct=100101.

Before operation register memory is:

```
00000000000000000000000000100101
00000000000000000000000001011101
00000000000000000000000010000010
00000000000000001010010111010111
00000000000000000000000010000010
10101010101010101010111111111111
00000000000000000000000000000000
```

Testbench:

```
PC:  2, instruction: 00000000100001010011000000100101
opcode: 000000, rs: 00100, rt: 00101 , rd=00110,dest=00110
 memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
Register: read_data_1: 00000000000000000000000010000010, read_data_2: 10101010101010101010111111111111, write_data: 10101010101010101010111111111111
```

After operation register memory is:

```
5   00000000000000000000000001011101
6   00000000000000000000000010000010
7   00000000000000001010010111010111
8   00000000000000000000000010000010
9   10101010101010101010111111111111
0   10101010101010101010111111111111
1   00000000000000000000000000000000
```

After or operation register conntent of rs (00100) and register content of rt(00101) is make or operation and result is writed in rd(00110).

Jump,Jal,Jr instructions:

This part in mips_32 is control jump,jal,jr operations.

```
always @(posedge clock) begin
  if(j==1)begin
    PC= jumpOut;
  end
  else if(andRes==1)begin
    PC = PC+1+extendedImmediate;
  end
  else if(jal) begin
    registers[31] = PC+1;
    PC=jumpOut;
    $writememb("registers.txt", registers);
  end
  else if(jr) begin
    PC=read_data_1;
  end
  else
    PC = PC+1;
end
```

Testbench of Jump :

Instruction is:

```
00001000000000000000000000000110
```

After jump operation,register is transfered 2 to 6.It is jumped that adress.

```
#
#  PC:  2, instruction: 00001000000000000000000000000110
#  opcode: 000010, rs: 00000, rt: 00000 , rd=00000,dest=00000
#   memtoReg =0 , regWrite =0 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=00, AluSrc=0, branch=0, j=1, jal=0, jr=0, regDest=0,clock= 1
#  Register: read_data_1: 00000000000000000000000000100101, read_data_2: 00000000000000000000000000100101, write_data: 00000000000000000000000001001010
#
#  PC:  2, instruction: 00001000000000000000000000000110
#  opcode: 000010, rs: 00000, rt: 00000 , rd=00000,dest=00000
#   memtoReg =0 , regWrite =0 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=00, AluSrc=0, branch=0, j=1, jal=0, jr=0, regDest=0,clock= 0
#  Register: read_data_1: 00000000000000000000000000100101, read_data_2: 00000000000000000000000000100101, write_data: 00000000000000000000000001001010
#
#  PC:  6, instruction: 00000001110000111010000000100100
#  opcode: 000000, rs: 01110, rt: 00011 , rd=10100,dest=10100
#   memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
#  Register: read_data_1: 10101010101010101010111111111111, read_data_2: 00000000000000001010010111010111, write_data: 00000000000000001010010111010111
#
#  PC:  6, instruction: 00000001110000111010000000100100
#  opcode: 000000, rs: 01110, rt: 00011 , rd=10100,dest=10100
#   memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 0
#  Register: read_data_1: 10101010101010101010111111111111, read_data_2: 00000000000000001010010111010111, write_data: 00000000000000001010010111010111
#
```

Testbench of jumpAndLink:

Instruction is:

```
00001000000000000000000000000101
```

Testbench:

```
PC:  2, instruction: 00001000000000000000000000000101
opcode: 000010, rs: 00000, rt: 00000 , rd=00000,dest=00000
 memtoReg =0 , regWrite =0 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=00, AluSrc=0, branch=0, j=1, jal=0, jr=0, regDest=0,clock= 0
Register: read_data_1: 00000000000000000000000000100101, read_data_2: 00000000000000000000000000100101, write_data: 00000000000000000000000001001010

PC:  5, instruction: 00000001001010010110000100000
opcode: 000000, rs: 01001, rt: 01010 , rd=01011,dest=01011
 memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
Register: read_data_1: 10000000000000000000000000000001, read_data_2: 00000000000000000000000000000000, write_data: 10000000000000000000000000000001
```

After jumpAndLink operation,register is transfered 2 to 5.It is jumped that adress.And Pc+1 assigned $31.

Testbench of beq:

Instruction is:

```
00010000000000010000000000000100
```

Note that rs and rt contents equal:

```
00000000000000000000000000100101
00000000000000000000000000100101
00000000000000000000000010000010
00000000000000001010010111010111
00000000000000000000000010000010
```

```
PC:  1, instruction: 00010000000000010000000000000100
opcode: 000100, rs: 00000, rt: 00001 , rd=00000,dest=00001
 memtoReg =0 , regWrite =0 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=01, AluSrc=0, branch=1, j=0, jal=0, jr=0, regDest=0,clock= 0
Register: read_data_1: 00000000000000000000000000100101, read_data_2: 00000000000000000000000000100101, write_data: 00000000000000000000000000000000

PC:  6, instruction: 00000001001010100101100000100000
opcode: 000000, rs: 01001, rt: 01010 , rd=01011,dest=01011
 memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
Register: read_data_1: 10000000000000000000000000000001, read_data_2: 00000000000000000000000000000000, write_data: 10000000000000000000000000000001
```

After beq operation,pc transfered new address.

Testbench of sh operation:

Instruction memory:

Opcode: 10 1001 , rs=00000, rt=00001 , immediate=16'b0

```
1   10100100000000010000000000000000
```

Register Memory:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/registers/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000000
5  11111100000001000000111010001001
6  00000000000000000010100111101110
7  00000100000101000010010010010001
8  00001000100001000010001000010000
9  00010001000100000100100000100100
```

Before sh operation Data Memory:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/datamem/data_memory
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000000
5  10000001000010001111111011101110
6  11010101010010101000010010101010
7  00000100010000100001000100001000
8  00000000000000000000000000000111
```

```
# Instruction: 10100100000000010000000000000000
# rsData: 00000000000000000000000000000000 rtData:11111100000001000000111010001001
# sh operation doing...
```

After sh operation,data memory:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/datamem/data_memory
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000111010001001
5  10000001000010001111111011101110
6  11010101010010101000010010101010
7  00000100010000100001000100001000
```

NEVRA GÜRSES
161044071

Testbench of sb operation:

Instruction memory:

Opcode: 10 1000 , rs=00000, rt=00001 , immediate=16'b0

```
1   10100000000000010000000000000000
```

Register Memory:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/registers/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000000
5  11111100000001000000111010001001
6  00000000000000000010100111101110
7  00000100000101000010010010010001
8  00000100010000100001000100001000
9  00010001000100000100100000100100
```

Before Data Memory:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/datamem/data_memory
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000000
5  10000001000010001111111011101110
6  11010101010010101000010010101010
7  00000100010000100001000100001000
8  00000000000000000000000000000111
```

```
# Instruction: 10100000000000010000000000000000
# rsData: 00000000000000000000000000000000 rtData:11111100000001000000111010001001
# sb operation doing...
```

After sb operation,data memory:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/datamem/data_memory
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000010001001
5  10000001000010001111111011101110
6  11010101010010101000010010101010
7  00000100010000100001000100001000
```

Testbench of sw operation:

Instruction memory:

Opcode: 10 1011 , rs=00000, rt=00001 , immediate=16'b0

```
1   10101100000000010000000000000000
```

NEVRA GÜRSES
161044071

Register Memory:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/registers/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000000
5  11111100000001000000111010001001
6  00000000000000000010100111101110
7  00000100000101000010010010010001
8  00000100010000100001000100001000
9  00010001000100000100100000100100
```

Before Data Memory:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/datamem/data_memory
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000000
5  10000010000100011111111011101110
6  11010101010010101000010010101010
7  00000100010000100001000100001000
8  00000000000000000000000000000111
```

```
# Instruction: 10101100000000010000000000000000
# rsData: 00000000000000000000000000000000 rtData:11111100000001000000111010001001
# sw operation doing...
```

After sw operation,data memory:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/datamem/data_memory
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  11111100000001000000111010001001
5  10000010000100011111111011101110
6  11010101010010101000010010101010
7  00000100010000100001000100001000
8  00000000000000000000000000000111
```

NEVRA GÜRSES
161044071

Testbench of lb operation:

Instruction memory:

Opcode: 10 0000 , rs=00001, rt=00010 , immediate=16'b0

```
1   10000000001000100000000000000000
```

Before writing register, register memory is:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/registers/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000011
5  00000000000000000000000000000001
6  00000000000000000000000000000000
7  00000100000101000010010010001
8  00000000000000000000000000000000
9  00010001000100000100100000100100
```

Data memory is:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips_data_memory_testbench/tb/data_memory
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  11111111111111111111111111110101
5  10000010000100011111111011101110
6  11010101010010101000010010101010
7  00000100010000100001000100001000
8  00000000000000000000000000000111
9  00000100000100010000100000100110
10 00000000000000000000000000000111
```

After perform lb operation, rt addresses(00010) content in register memory is changed.

R[rt]={24'b0,M[rs+ZeroExtImm](7:0)}

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/registers/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000011
5  00000000000000000000000000000001
6  00000000000000000000000011101110
7  00000100000101000010010010001
8  00000000000000000000000000000000
9  00010001000100000100100000100100
```

You can see that last 8 bit of 1. address of data memory is writed in 2. address of register memory.So lb is working true.

NEVRA GÜRSES
161044071

Testbench of lw operation:

Instruction memory:

Opcode: 10 0011 , rs=00000, rt=00100 , immediate=16'b0

```
1  10001100000001000000000000000000
```

Before writing register, register memory is:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/registers/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000011
5  00000000000000000000000000000001
6  00000000000000000000000011101110
7  00000100000101000010010010010001
8  00000000000000000000000000000000
9  00010001000100000100100000100100
```

Data memory is:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips_data_memory_testbench/tb/data_memory
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  11111111111111111111111111110101
5  10000010000100011111111011101110
6  11010101010010101000010010101010
7  00000100010000100001000100001000
8  00000000000000000000000000000111
9  00000100000100010000100000100110
10 00000000000000000000000000000111
```

After perform lw operation, rt addresses(00100) content in register memory is changed.

R[rt]= M[rs+SignExtImm]

Register memory after operation:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/registers/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000011
5  00000000000000000000000000000001
6  00000000000000000000000011101110
7  00000100000101000010010010010001
8  00000100010000100001000100001000
9  00010001000100000100100000100100
10 00000000000000000000000010010011
```

NEVRA GÜRSES
161044071

Testbench of lh operation:

Instruction memory:

Opcode: 10 0001 , rs=00000, rt=00001 , immediate=16'b0

```
1   10000100000000010000000000000000
```

Register memory before writing :

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/registers/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000000
5  00000000000000000000000000000000
6  00000000000000000000000000011101110
7  00000100000101000010010010010001
8  00000100010000100001000100001000
9  00010001000100000100100000100100
```

Data memory:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips_data_memory_testbench/tb/data_memory
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  11111111111111111111111111110101
5  10000001000010001111111011101110
6  11010101010010101000010010101010
7  00000100010000100001000100001000
8  00000000000000000000000000000111
```

After perform lh operation, rt addresses(00001) content in register memory is changed.

R[rt]={16'b0,M[rs+ZeroExtImm](15:0)}

Register memory after operation:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/registers/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000000
5  00000000000000001111111111110101
6  00000000000000000000000011101110
7  00000100000101000010010010010001
8  00000100010000100001000100001000
```

NOT: lhu operation is working same with lh out of it has different opcode and it makes sign extended immediate.

NEVRA GÜRSES
161044071

Testbench of lui operation:

Instruction Memory:

```
1  00111100001000001111001001000101
2
```

Register Memory Before Lui Operation:

```
1  // memory data file (do not edit the following line - required for mem load use)
2  // instance=/mips32_testbench/tb/registers/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
4  00000000000000000000000000000000
5  11111100000001000000111010001001
6  00000000000000000000101001111011l0
7  00000100000101000010010010010001
8  00000100010000100001000100001000
```

```
VSIM 446> step -current
# Writing data:11110010010001010000000000000000
# Instruction: 00111100001000001111001001000101
# rsData: 11111100000001000000111010001001 rtData:00000000000000000000000000000000
```

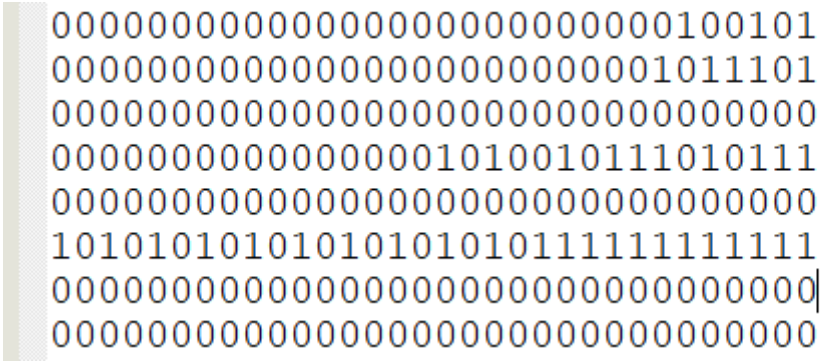
Register Memory After Lui Operation:

```
1  // memory data file (do not edit the following line - required
2  // instance=/mips32_testbench/tb/registers/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperl
4  11110010010001010000000000000000
5  11111100000001000000111010001001
6  00000000000000000000101001111011l0
7  00000100000101000010010010010001
```

NEVRA GÜRSES
161044071

Testing and,or,add operations together:

Before operations,data memory is:




```
SIM 40> step -current
# PC:  0, instruction: 00000000000000010001000000100000
# opcode: 000000, rs: 00000, rt: 00001 , rd=00010,dest=00010
#  memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 0
# Register: read_data_1: 00000000000000000000000000100101, read_data_2: 00000000000000000000000001011101, write_data: 00000000000000000000000010000010
#
# PC:  1, instruction: 00000000100001100100000000100100
# opcode: 000000, rs: 00010, rt: 00011 , rd=00100,dest=00100
#  memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
# Register: read_data_1: 00000000000000000000000010000010, read_data_2: 00000000000000001010010111010111, write_data: 00000000000000000000000010000010
#
# PC:  1, instruction: 00000000100001100100000000100100
# opcode: 000000, rs: 00010, rt: 00011 , rd=00100,dest=00100
#  memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 0
# Register: read_data_1: 00000000000000000000000010000010, read_data_2: 00000000000000001010010111010111, write_data: 00000000000000000000000010000010
#
# PC:  2, instruction: 00000000100001010011000000100101
# opcode: 000000, rs: 00100, rt: 00101 , rd=00110,dest=00110
#  memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
# Register: read_data_1: 00000000000000000000000010000010, read_data_2: 10101010101010101010111111111111, write_data: 10101010101010101010111111111111
#
# PC:  2, instruction: 00000000100001010011000000100101
# opcode: 000000, rs: 00100, rt: 00101 , rd=00110,dest=00110
#  memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 0
# Register: read_data_1: 00000000000000000000000010000010, read_data_2: 10101010101010101010111111111111, write_data: 10101010101010101010111111111111
#
#
#  PC:  3, instruction: 10001100111010000000000000000000
#  opcode: 100011, rs: 00111, rt: 01000 , rd=00000,dest=01000
#   memtoReg =1 , regWrite =1 , memRead =1 , memWrite =0 , extended =1 , luiControl =0 , load =01, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0,clock= 1
#  Register: read_data_1: 00000000000000000000000000000000, read_data_2: 10101011111111111000011111010111, write_data: 10101011111111111000011111010111
#
#  PC:  3, instruction: 10001100111010000000000000000000
#  opcode: 100011, rs: 00111, rt: 01000 , rd=00000,dest=01000
#   memtoReg =1 , regWrite =1 , memRead =1 , memWrite =0 , extended =1 , luiControl =0 , load =01, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0,clock= 0
#  Register: read_data_1: 00000000000000000000000000000000, read_data_2: 10101011111111111000011111010111, write_data: 10101011111111111000011111010111
#
```

After add,or,and operation together,register memory is:

```
// format=bin addressradix=h dataradix=b
00000000000000000000000000100101
00000000000000000000000001011101
00000000000000000000000010000010
00000000000000001010010111010111
00000000000000000000000010000010
10101010101010101010111111111111
10101010101010101010111111111111
00000000000000000000000000000000
```

Instuction Memory is:

```
1  00000000000000010001000000100000
2  00000000100001100100000000100100
3  00000001000010100110000000100101
4  10001100111010000000000000000000
5  00000001001010100101100000100000
6  00000001110000111010000000100100
7  00000001100010101110000000100101
```

Before changing,register memory is:

```
1  // memory data file (do not edit the following line - required for mem lo
2  // instance=/mips32_testbench/tb/register/registers
3  // format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noadd
4  00000000000000000000000000100101
5  00000000000000000000000001011101
6  00000000000000000000000000000000
7  00000000000000001010010111010111
8  00000000000000000000000000000000
9  10101010101010101010111111111111
10 00000000000000000000000000000000
11 00000000000000000000000000000000
12 00000000000000000000000000000000
13 10000000000000000000000000000001
14 00000000000000000000000000000000
15 10000000000000000000000000000001
16 00000000000000000000000001000100
17 00000000000000000000000010001010
18 00000000000000000000000000000000
19 00000000000000000000000000000000
20 00000000000000000000000000010001
```

NEVRA GÜRSES
161044071

TestBench:

```
#  PC:  0, instruction: 00000000000000010001000000100000
#  opcode: 000000, rs: 00000, rt: 00001 , rd=00010,dest=00010
#   memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 0
#  Register: read_data_1: 00000000000000000000000000100101, read_data_2: 00000000000000000000000001011101, write_data: 00000000000000000000000010000010
#
#  PC:  1, instruction: 00000000010000110010000000100100
#  opcode: 000000, rs: 00010, rt: 00011 , rd=00100,dest=00100
#   memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
#  Register: read_data_1: 00000000000000000000000010000010, read_data_2: 00000000000000010100101111010111, write_data: 00000000000000000000000010000010
#
#  PC:  1, instruction: 00000000010000110010000000100100
#  opcode: 000000, rs: 00010, rt: 00011 , rd=00100,dest=00100
#   memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 0
#  Register: read_data_1: 00000000000000000000000010000010, read_data_2: 00000000000000010100101111010111, write_data: 00000000000000000000000010000010
#
#  PC:  2, instruction: 00000000100001010011000000100101
#  opcode: 000000, rs: 00100, rt: 00101 , rd=00110,dest=00110
#   memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
#  Register: read_data_1: 00000000000000000000000010000010, read_data_2: 10101010101010101011111111111111, write_data: 10101010101010101011111111111111
#
#  PC:  2, instruction: 00000000100001010011000000100101
#  opcode: 000000, rs: 00100, rt: 00101 , rd=00110,dest=00110
#   memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 0
#  Register: read_data_1: 00000000000000000000000010000010, read_data_2: 10101010101010101011111111111111, write_data: 10101010101010101011111111111111
#
#  PC:  3, instruction: 10001100111010000000000000000000
```

```
PC:  3, instruction: 10001100111010000000000000000000
opcode: 100011, rs: 00111, rt: 01000 , rd=00000,dest=01000
 memtoReg =1 , regWrite =1 , memRead =1 , memWrite =0 , extended =1 , luiControl =0 , load =01, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0,clock= 1
Register: read_data_1: 00000000000000000000000000000000, read_data_2: 10101011111111111000011111010111, write_data: 10101011111111111000011111010111

PC:  3, instruction: 10001100111010000000000000000000
opcode: 100011, rs: 00111, rt: 01000 , rd=00000,dest=01000
 memtoReg =1 , regWrite =1 , memRead =1 , memWrite =0 , extended =1 , luiControl =0 , load =01, AluOp=00, AluSrc=1, branch=0, j=0, jal=0, jr=0, regDest=0,clock= 0
Register: read_data_1: 00000000000000000000000000000000, read_data_2: 10101011111111111000011111010111, write_data: 10101011111111111000011111010111

PC:  4, instruction: 00000001001010100101100000100000
opcode: 000000, rs: 01001, rt: 01010 , rd=01011,dest=01011
 memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
Register: read_data_1: 10000000000000000000000000000001, read_data_2: 00000000000000000000000000000000, write_data: 10000000000000000000000000000001

PC:  4, instruction: 00000001001010100101100000100000
opcode: 000000, rs: 01001, rt: 01010 , rd=01011,dest=01011
 memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 0
Register: read_data_1: 10000000000000000000000000000001, read_data_2: 00000000000000000000000000000000, write_data: 10000000000000000000000000000001

PC:  5, instruction: 00000001110000111010000000100100
opcode: 000000, rs: 01110, rt: 00011 , rd=10100,dest=10100
 memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
Register: read_data_1: 10101010101010101011111111111111, read_data_2: 00000000000000010100101111010111, write_data: 00000000000000010100101111010111

PC:  5, instruction: 00000001110000111010000000100100
opcode: 000000, rs: 01110, rt: 00011 , rd=10100,dest=10100
 memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 0
Register: read_data_1: 10101010101010101011111111111111, read_data_2: 00000000000000010100101111010111, write_data: 00000000000000010100101111010111
```

```
PC:  6, instruction: 00000000110001010111000000100101
opcode: 000000, rs: 00110, rt: 00101 , rd=01110,dest=01110
 memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 1
Register: read_data_1: 10101010101010101011111111111111, read_data_2: 10101010101010101011111111111111, write_data: 10101010101010101011111111111111

PC:  6, instruction: 00000000110001010111000000100101
opcode: 000000, rs: 00110, rt: 00101 , rd=01110,dest=01110
 memtoReg =0 , regWrite =1 , memRead =0 , memWrite =0 , extended =0 , luiControl =0 , load =00, AluOp=10, AluSrc=0, branch=0, j=0, jal=0, jr=0, regDest=1,clock= 0
Register: read_data_1: 10101010101010101011111111111111, read_data_2: 10101010101010101011111111111111, write_data: 10101010101010101011111111111111
```

NEVRA GÜRSES
161044071

After operations,register memory is:

```
// memory data file (do not edit the following line - required for mem load us
// instance=/mips32_testbench/tb/register/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000100101
00000000000000000000000001011101
00000000000000000000000010000010
00000000000000001010010111010111
00000000000000000000000010000010
10101010101010101010111111111111
10101010101010101010111111111111
00000000000000000000000000000000
10101011111111111000011111010111
10000000000000000000000000000001
00000000000000000000000000000000
10000000000000000000000000000001
00000000000000000000000001000100
00000000000000000000000010001010
10101010101010101010111111111111
00000000000000000000000000000000
00000000000000000000000000010001
```

NOTE:My processor performs,add,or,and,beq,j,jal,jr,lw,lb,lbu,lh,lhu,sw,sb,sh operations.