COMPUTER ORGANIZATION FINAL PROJECT REPORT :

*AUTHOR : lemye ceren gumus – 151044071*

Instructions that is implemented: store word, store byte, store halfword, andi, ori ,xori ,addu, and, jr, nor, or, slti, sltiu, lbu, lhu, load linked, slt, sltu, srl, sra, sub, subu jump, jal addi, addiu, and, beq, bneq, ll, lui, lw, ori, slti , sltui, sw,xor,add signed,substract signed,sustract unsigned

Top Level Design File: mips\_core

File: mips\_testbench

mips\_registers.v is register module

registers.mem for initialization of registers

mips\_instr\_mem.v is instruction memory module

instruction.mem for instruction memory initialization

mips\_data\_mem.v is data memory module

data.mem for data memory initialization

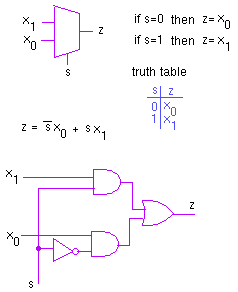
Before running the program in modelsim, you need to put the data.mem, registers.mem and instruction.mem files under the simulaiton / modelsim folder.

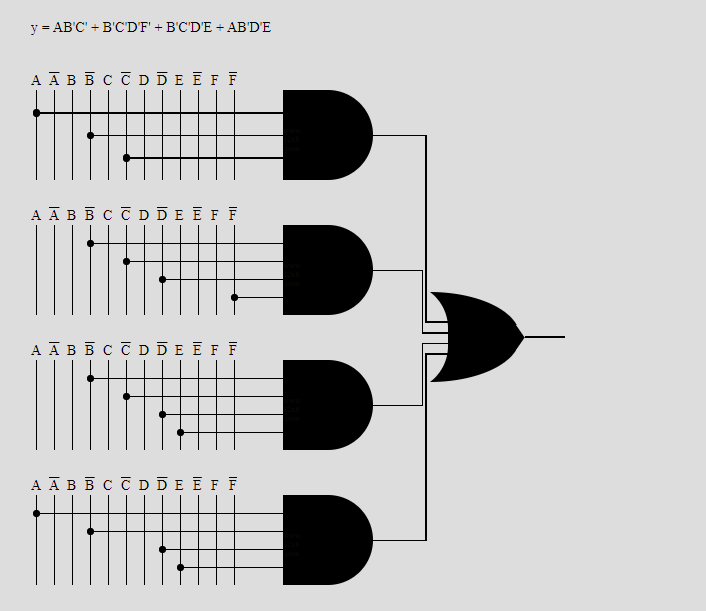
After running the program in modelsim res\_registers.mem and res\_data.mem two files are created under the simulaiton / modelsim folder.

The res\_registers.mem file contains the current registers after the program runs.

res\_data.mem contains the current version of memory after running the program.

*Firslty ,it is created control unit module according to behaviurol code,*





|  |
| --- |
|  |
|  |  |
|  | if(opCode == 6'b000000)begin //R-Type |
|  | if (functCode == 6'b001000)begin // jr |
|  | regWr = 1'b0; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  |  |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |
|  |
|  |
|  | regWr = 1'b1; |
|  | regDest = 1'b1; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  | end |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  |  |
|  | else if (opCode == 6'b001100)begin //andi |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  | end |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if (opCode == 6'b001010)begin //slti |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  | end |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if (opCode == 6'b001011)begin //sltiu |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  |  |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if (opCode == 6'b100011)begin //lw |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b1; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  |  |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if (opCode == 6'b100100)begin //lbu |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b1; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  | end |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if (opCode == 6'b100101)begin //lhu |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b1; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  |  |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if(opCode == 6'b110000)begin //ll |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b1; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  | end |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  | end |
|  | end |
|  | else if (opCode == 6'b101011)begin //sw |
|  | regWr = 1'b0; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b1; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  | end |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  | end |
|  | end |
|  | else if (opCode == 6'b101000)begin //sb |
|  | regWr = 1'b0; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b1; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  | end |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if (opCode == 6'b101001)begin //sh |
|  | regWr = 1'b0; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b1; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  | end |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if (opCode == 6'b001000)begin //addi |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  |  |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if (opCode == 6'b001001)begin //addiu |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  | end |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if(opCode == 6'b001101)begin //ori |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  | end |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if(opCode == 6'b001110)begin //xori |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  |  |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if(opCode == 6'b000100)begin //beq |
|  | regWr = 1'b0; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  |  |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if(opCode == 6'b000101)begin //bne |
|  | regWr = 1'b0; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  | end |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |  |
|  |  |
|  | else if(opCode == 6'b000010)begin //jump |
|  | regWr = 1'b0; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  |  |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |
|  |
|  |
|  | else if(opCode == 6'b000011)begin //jal |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  |  |
|  | else if(regDest == 0)begin |
|  | rd = 5'b11111; |
|  |
|  |
|  | else if(opCode == 6'b001111)begin //lui |
|  | regWr = 1'b1; |
|  | regDest = 1'b0; |
|  | memRead = 1'b0; |
|  | memWr = 1'b0; |
|  | if(regDest == 1)begin |
|  | rd = instruction[15:11]; //rd |
|  |
|  | else if(regDest == 0)begin |
|  | rd = instruction[20:16]; |

*This shows that structural Verilog code,*

*After that mips\_core module is created,*

*if instructions are written data.mem file e.g store store word etc,mips\_core have to be controls these.*

*if instructions are written registers.mem file e.g RTYPE INSTRUCTIONS AND LW LHU etc etc,mips\_core have to be controls these.*

*I\_TypeInstrAssign module should write each instruction results aluresult or writedata.*

*ALURESULT changes the registers.mem*

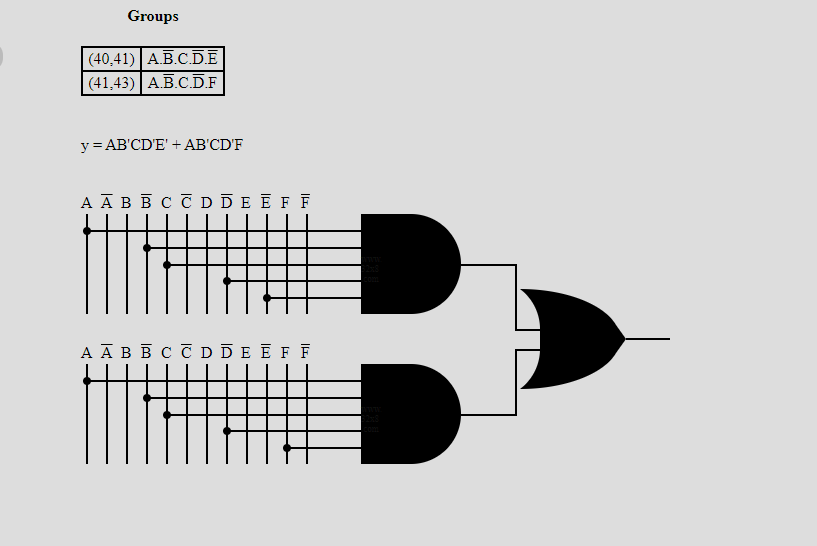
*Writedata changes the data.mem.*

*Instructions is created according to their tasks.*

The following simulation circuits serve to identify instructions with specific funccode and opcode.

*Store instructions designs according to opcode*

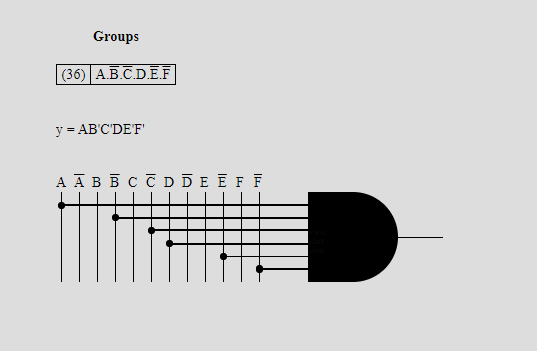
|  |
| --- |
| if(opCode==43)begin |
|  |
|  | writeData = rtContent; |
|  | zero = 1'b0; |
|  | End |
|  | else if (opCode == 40)begin // Store Byte |
|  | writeData = rtContent; |
|  | zero = 1'b0; |
|  | End |
|  | else if (opCode == 41)begin // Store Halfword |
|  | writeData = rtContent; |
|  | zero = 1'b0; |
|  | End |



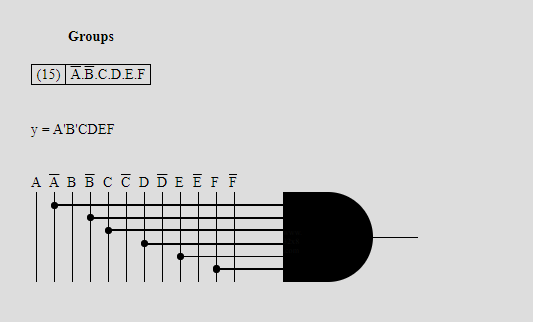
*It shows that these instructions are store instructions.*

*LBU instructions designs according to opcode*

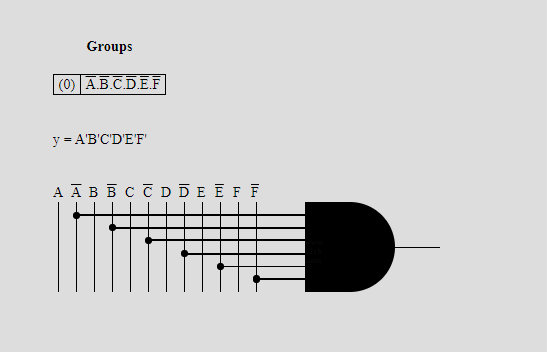
|  |
| --- |
| if(opCode==36)begin |
|  | writeData = {24'b0,data\_mem[address][7:0]}; |
|  | zero = 1'b0; |
|  | end |



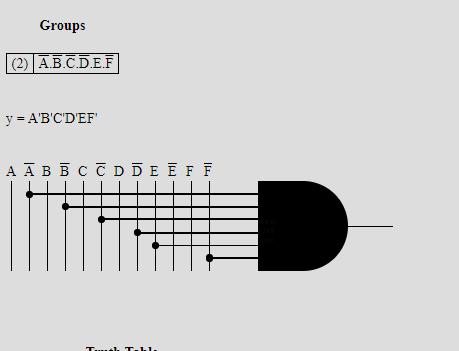
*LUI instructions designs according to opcode*



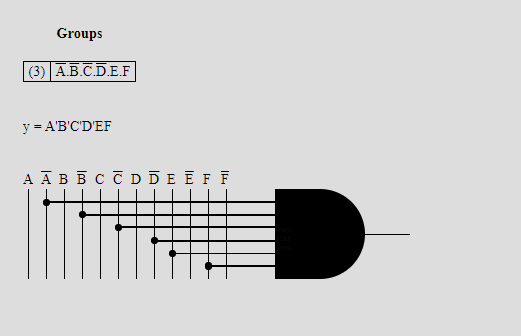
*SLL instructions designs according to opcode*



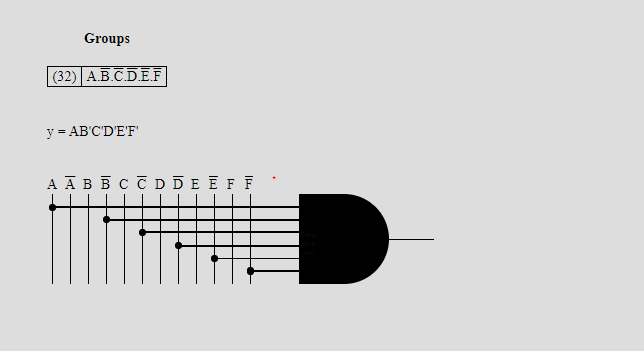
*SRL instructions designs according to opcode*



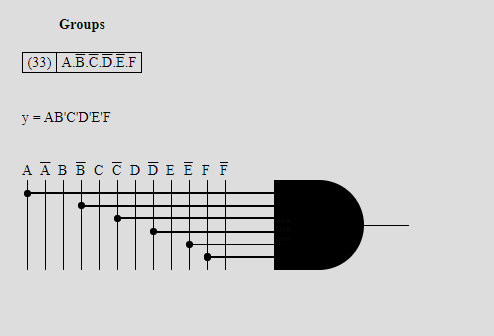
*SRA instructions designs according to opcode*



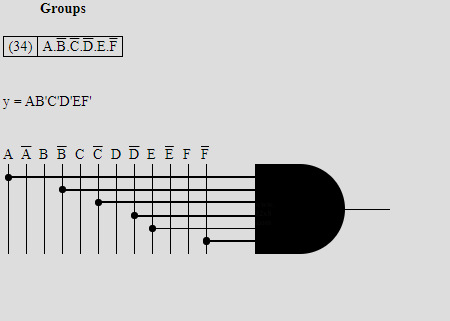
*ADDSIGNED instructions designs according to opcode*



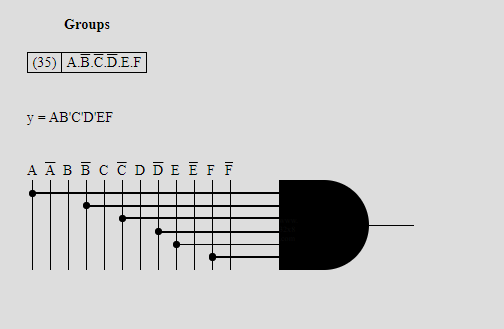
*ADDUNSIGNED instructions designs according to opcode*



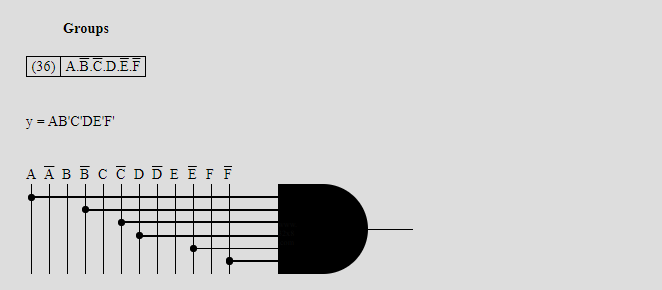
*SUBSTRACTSIGNED instructions designs according to opcode*



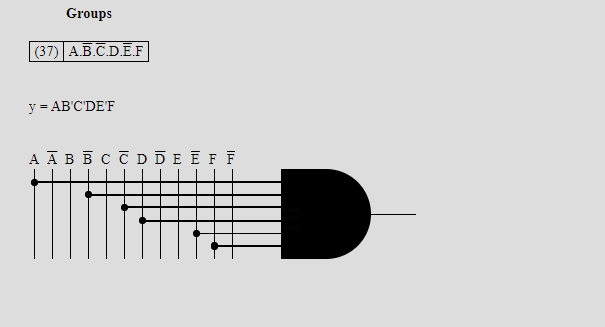
*SUBSTRACTUNSIGNED instructions designs according to opcode*



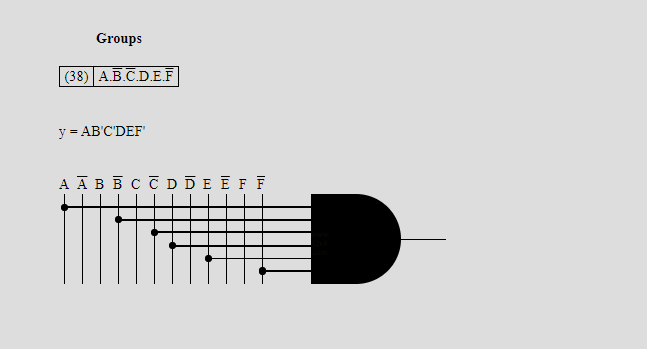
*AND instructions designs according to opcode*



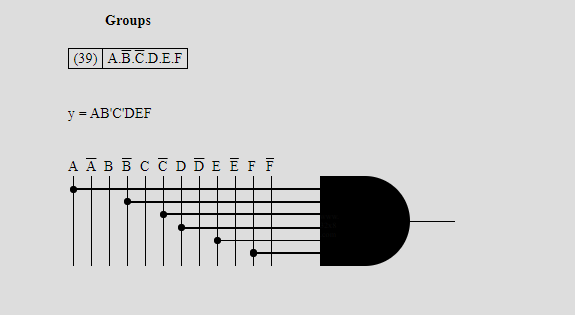
*OR instructions designs according to opcode*



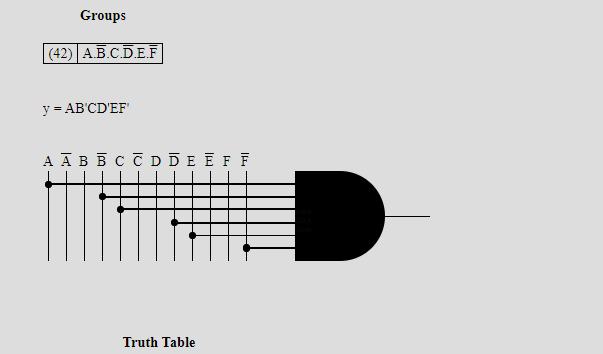
*XOR instructions designs according to opcode*



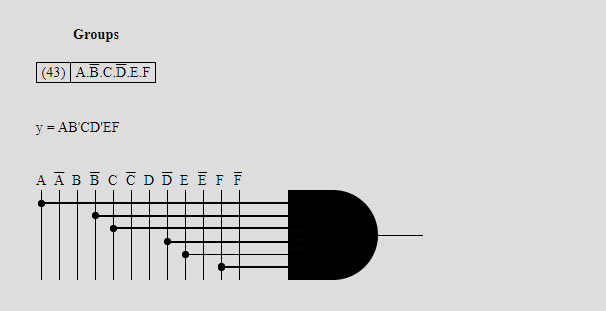
*NOR instructions designs according to opcode*



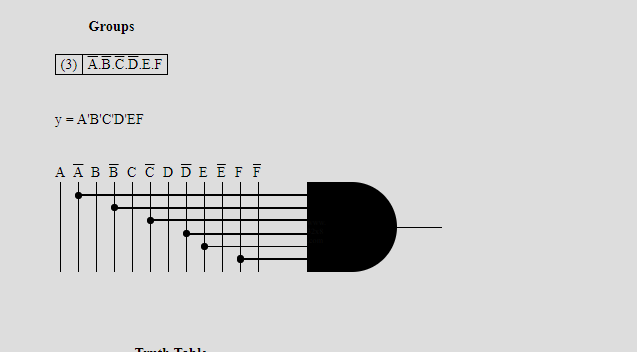
*SLT instructions designs according to opcode*



*SLTU instructions designs according to opcode*

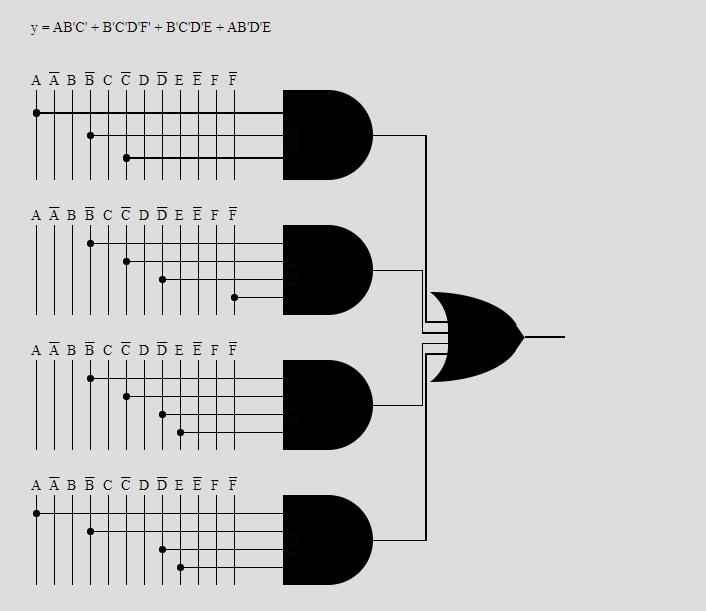


*JAL instructions designs according to opcode*

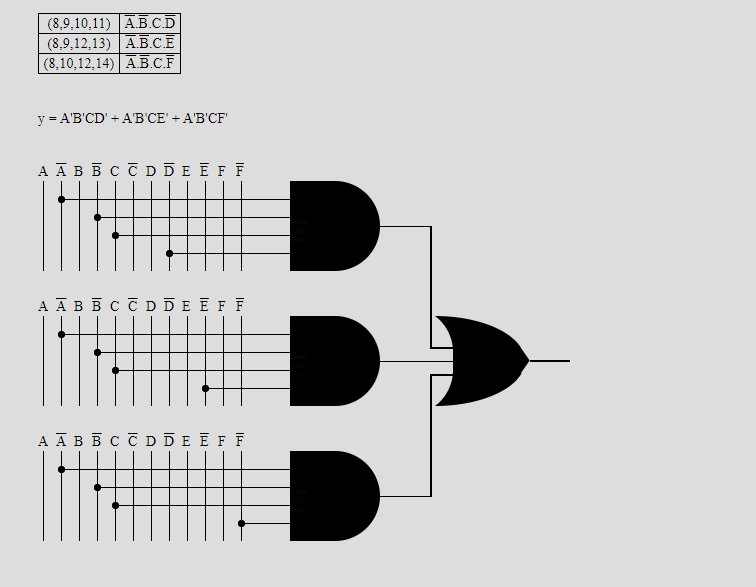


*DETEMINED ALURESULT AND FUNCODE HAVE SPECIFIC OPERATIONS.*

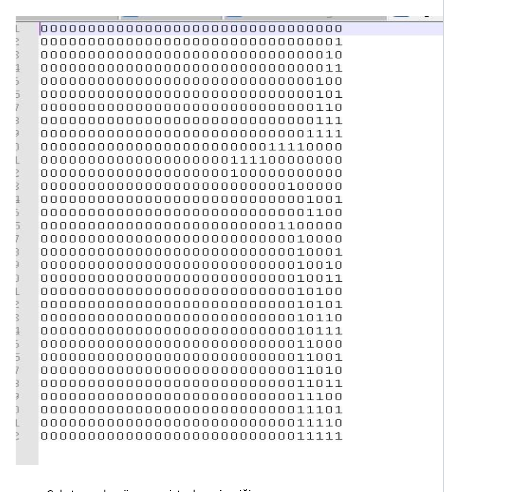
*SPECIFIC FUNCCODE*



*SPECIFIC ALURESULT*



*Results:*



*Module Itype\_instructionresults shows that all instructions results:*

