NEVRA GÜRSES

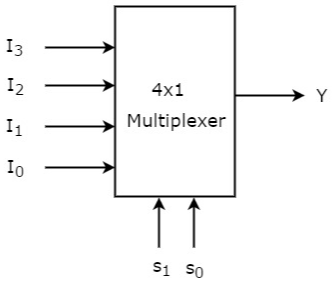
161044071

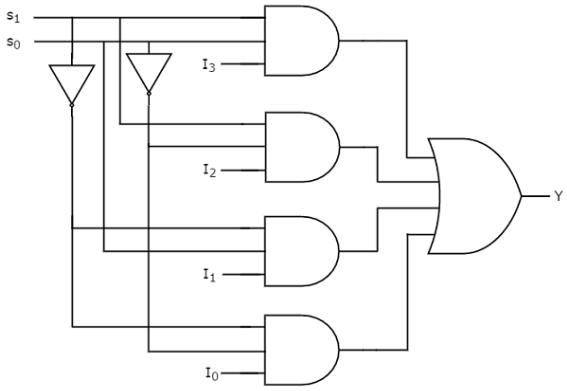
COMPUTER ORGANIZATION PROJECT -2

The verilog modules in my Project:

module mux (muxOut ,I0,I1,I2,I3,s0,s1) : This module creates 4:1 multiplexer.

It takes 2 selection bit and 4 bit input. It gives 1 output (I0 or I1 or I2 or I3).It is like that:





The output of multiplexer is : s0’s1’I0 + s0s1’I1 + s0’s1I2 + s0s0I4

The testbench of mux module:

```
# Loading work.mux_testbench
# Loading work.mux
add wave -position insertpoint  \
sim:/mux_testbench/I0 \
sim:/mux_testbench/I1 \
sim:/mux_testbench/I2 \
sim:/mux_testbench/I3 \
sim:/mux_testbench/muxOut \
sim:/mux_testbench/s0 \
sim:/mux_testbench/s1
VSIM 22> step -current
# I0 =1, I1=0, I2=0, I3=0, s1=0 s0=0  muxOut=1
# I0 =0, I1=1, I2=0, I3=0, s1=0 s0=1  muxOut=1
# I0 =0, I1=0, I2=1, I3=0, s1=1 s0=0  muxOut=1
# I0 =0, I1=0, I2=0, I3=1, s1=1 s0=1  muxOut=1
```

I tested all cases.Firstly ,I give s1=0 and s0=0, this give I0 as result.If only I0 is 1 others is zero result is 1 .

Secondly ,I give s1=0 and s0=1, this give I1 as result.If only I1 is 1 others is zero result is 1 .

Thirdly ,I give s1=1 and s0=0, this give I2 as result.If only I2 is 1 others is zero result is 1 .

Fourthly ,I give s1=1 and s0=1, this give I3 as result.If only I3 is 1 others is zero result is 1 .

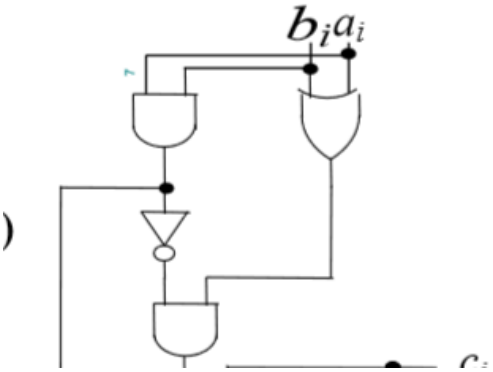
This multiplexer is using for selection of ALU operation according to ALUopcode.

NEVRA GÜRSES

161044071

module xorGate (result,a ,b): This module creates xor gate. In our lesson slides

a xor b = (a or b) and (not (a and b)).This xor gate is using in ALU.Because Alu includes full adder and full adder using this xor gate.This xor gate is :



Testbench of xorGate module:

```
# Loading work.xorGate_testbench
# Loading work.xorGate
add wave -position insertpoint  \
sim:/xorGate_testbench/a \
sim:/xorGate_testbench/b \
sim:/xorGate_testbench/result
VSIM 34> step -current
# a =0, b=0, result=0
# a =1, b=0, result=1
# a =0, b=1, result=1
# a =1, b=1, result=0

VSIM 35>
```

If input a =0, b =0 then a xor b = 0

If input a =1 ,b =0 then a xor b = 1

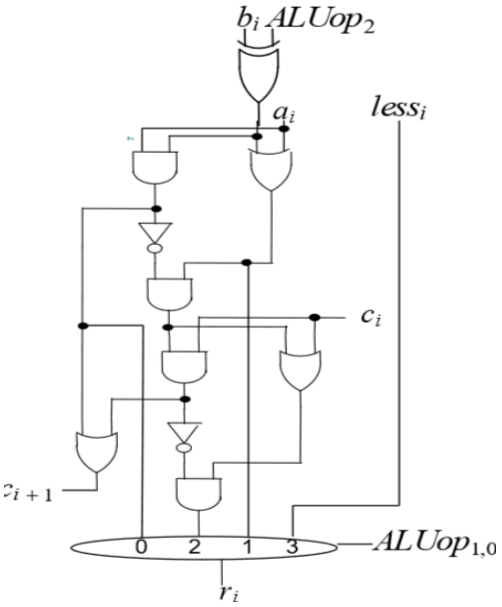
If input a =0 b =1 then a xor b = 1

If input a =1 b =1 then a xor b = 0

NEVRA GÜRSES

161044071

module Alu_1Bit(carryOut,outputAlu,a,b,carryIn,less,AluOp): This module for creating 1 bit Alu. In olur lessons this ALU is :



For creating this Alu ,firstly I call xor gate module for selection = b xor ALUop ,after that I call this function secondly and thirdly.Because in this circuit :

(a or selection) and (not (a and selection)) is using and this is (a xor selection).After that this result again xor with carry-in and so addition or subtruct part of multiplexer is created .After I create carry-out of Alu. After that and ,or ,set-on-less- than parts is fixing multiplexer input.And I call mux module for selection of ALU operation.

Testbench of 1-bit-Alu module:

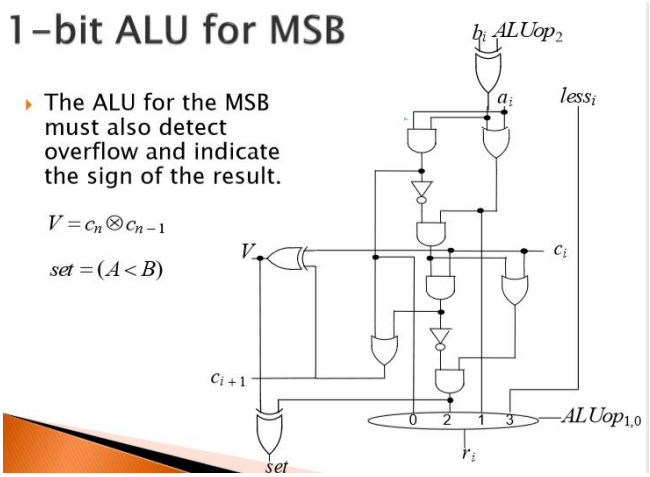
```
VSIM 35> vsim work.Alu_1Bit_testbench
# vsim work.Alu_1Bit_testbench
# Loading work.Alu_1Bit_testbench
# Loading work.Alu_1Bit
# Loading work.xorGate
# Loading work.mux
add wave -position insertpoint  \
sim:/Alu_1Bit_testbench/AluOp \
sim:/Alu_1Bit_testbench/a \
sim:/Alu_1Bit_testbench/b \
sim:/Alu_1Bit_testbench/carryIn \
sim:/Alu_1Bit_testbench/carryOut \
sim:/Alu_1Bit_testbench/less \
sim:/Alu_1Bit_testbench/outputAlu
VSIM 37> step -current
#  AluOp=000 , a =1, b=1, carryIn=0, less=0, outputAlu=1, carryOut=1
#  AluOp=001 , a =0, b=1, carryIn=1, less=0, outputAlu=1, carryOut=1
#  AluOp=010 , a =1, b=1, carryIn=1, less=0, outputAlu=1, carryOut=1
#  AluOp=110 , a =1, b=0, carryIn=0, less=0, outputAlu=0, carryOut=1
#  AluOp=111 , a =0, b=0, carryIn=0, less=1, outputAlu=1, carryOut=0

VSIM 38>
```

In testbench of 1-bit-alu module : If Aluop is 000 then ALU makes and operation.If Aluop is 001 then ALU makes or operation. If Aluop is 010 then ALU makes addition operation. If Aluop is 110 then ALU makes subtruction operation. If Aluop is 111 then ALU makes set-on-less-than operation. Inputs is a,b ,carryIn and less.

NEVRA GÜRSES

161044071

module MSB_1BitAlu(carryOut,outputAlu,V,set,a,b,carryIn,less,AluOp); This module for creating most significant bit of Alu.I created below circuit.



It is similar with 1 bit alu.But it has some differences.It has two more outputs that are V and set. V is detect overflow occurence.And set is used for set-on-less-than (A<B) operations.

Testbench of this module:

```
n:/MSB_1Bit_testbench/carryOut \
n:/MSB_1Bit_testbench/carryIn \
n:/MSB_1Bit_testbench/b \
n:/MSB_1Bit_testbench/a \
n:/MSB_1Bit_testbench/V \
n:/MSB_1Bit_testbench/AluOp
M 25> step -current
 AluOp=000 , a =1, b=1, carryIn=0, less=0,  V=1, Set=1,carryOut=1 outputAlu=1,
 AluOp=001 , a =0, b=1, carryIn=1, less=0,  V=0, Set=0,carryOut=1 outputAlu=1,
 AluOp=010 , a =1, b=1, carryIn=1, less=0,  V=0, Set=1,carryOut=1 outputAlu=1,
 AluOp=110 , a =1, b=0, carryIn=0, less=0,  V=1, Set=1,carryOut=1 outputAlu=0,
 AluOp=111 , a =0, b=0, carryIn=0, less=1,  V=0, Set=1,carryOut=0 outputAlu=1,

M 26>
```

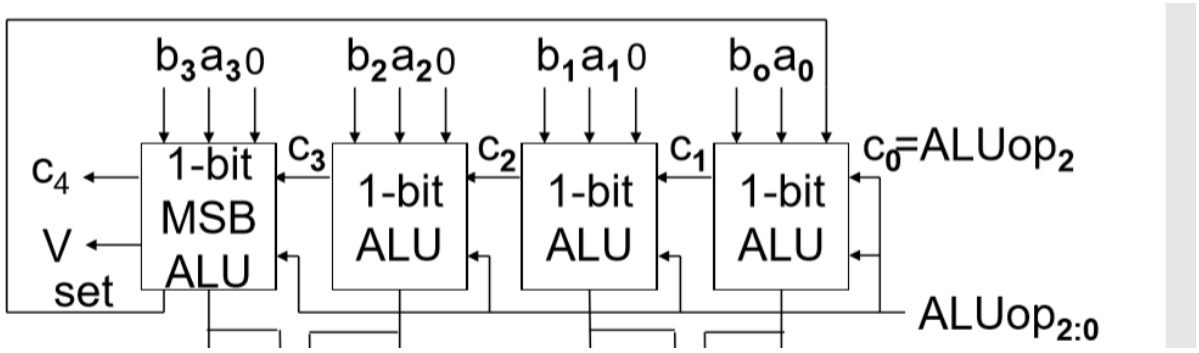
If a=1, b=1, cin=1 then addition result =1 and cout=1. V= (cin xor cout) so V is (1 xor 1)=0

Set is (V xor addition result) . So set = 0 xor 1 and set is 1. =>This is an explanation of testbench when AluOp =010.

NEVRA GÜRSES

161044071

module Alu_32_Bit (carryOut,outputAlu,a,b,carryIn,less,AluOp): This module for creating 32-bit-Alu. It uses 1 bit Alu module and most significant 1 bit Alu module. This module uses 31 times 1- bit- Alu module and last bit is most significant bit Alu.I created below circuit in our slides for 32 bit Alu:



For set-on-less than part :set output of MSB Alu is goes less bit part of first Alu.And less parts of other Alu's is 0. c0 is AluOp2.

Testbench of this module:

```
sim:/Alu_32_Bit_testbench/carryOut \
sim:/Alu_32_Bit_testbench/carryIn \
sim:/Alu_32_Bit_testbench/b \
sim:/Alu_32_Bit_testbench/a \
sim:/Alu_32_Bit_testbench/AluOp
VSIM 32> step -current
#  AluOp=000,a=11101101010110101011010101101010,b=01011110111111011111101111110111,cin=0,outputAlu=01001100010110001011000101100010,cOut=1
#  AluOp=001,a=10101010101010101010101010101010,b=01100011011000110110001101100011,cin=0,outputAlu=11101011111010111110101111101011,cOut=1
#  AluOp=010,a=00101010001010100010101000101010,b=00101010001010100010101000101010,cin=0,outputAlu=01010100010101000101010001010100,cOut=0
#  AluOp=110,a=01111010011110100111101001111010,b=00101010001010100010101000101010,cin=0,outputAlu=01010000010100000101000001010000,cOut=1
#  AluOp=111,a=10100000000000000000000000000000,b=10110000000000000000000000000000,cin=0,outputAlu=00000000000000000000000000000001,cOut=0

VSIM 33>
```

In testbench of 32-bit-alu module : If Aluop is 000 then ALU makes and operation.If Aluop is 001 then ALU makes or operation. If Aluop is 010 then ALU makes addition operation. If Aluop is 110 then ALU makes subtruction operation. If Aluop is 111 then ALU makes set-on-less-than operation.

For example when ALUOP 111 and "a" is smaller than "b" , in my testbench result is 1.Set on less than part of Alu works.

NEVRA GÜRSES

161044071

Logic gates I used for the 1-Bit-ALU :

7 gate for multiplexer.

In Alu :12 gate for 3 times xor gate (each xor gate includes 4 gate) calling,3 for carr-out finding, 2 for “or” and “and“ part of multiplexer.

Total = 12 + 3 + 7 + 2 = 24 gates.

Logic gates I used for the 32-Bit-ALU :

For most significant 1 bit alu module: 24 (same in 1 bit alu) + 8 (two times xor module calling) = 32 gates.

For 32-bit-Alu:

(24*31) + 32= 776 gate.