ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION PROJECT 1 REPORT

PROJECT NO	:	1
PROJECT DATE	:	11.05.2022
GROUP NO	:	G29

GROUP MEMBERS:

150190091	:	Cemalettin Celal TOY
150190087	:	Baran Işık
150190084	:	Mert Can ARABACI

SPRING 2022

Contents

1 INTRODUCTION

In this project we will create a basic computer organization part by part. First we will implement necessary circuit elements such registers and ALU. Then we will combine them with the help of multiplexers and create the system.

2 IMPLEMENTATIONS OF EACH PART

Part 1

Inputs and Outputs of the register:

- Inputs:
  - E is enable input, when it is 0, output of the register will not change no matter what you give as input.
  - FunSel is a 2 bit input that decides which operation performs in the register.
  - I is the input number that will be loaded to the register. It is 8 bit for 8 bit register and 16 bit for 16 bit register.
- Outputs:
  - Q is the output of the register. It is 8 bit for 8 bit register and 16 bit for 16 bit.

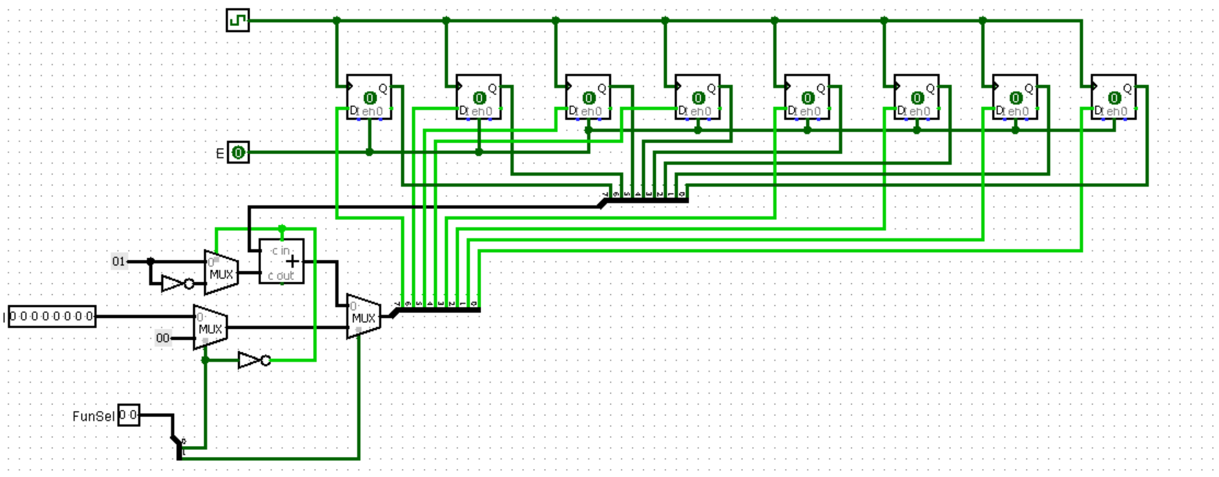
To implement 8 bit register, we used 8 1 bit D flip flops, 3 8 bit 2:1 Multiplexer and a 8 bit adder/substractor. Adder/substractor decrements the value in the register if Funsel is equal to 00 and increments if Funsel is equal to 01.

Control inputs

- FunSel - Each bit of FunSel are selection bits for different Multiplexers and FunSel[0] determines the Cin of adder/substractor.
  - 00 : Decrement the Q value of register. Since FunSel[0] is 0, Cin is 1 and multiplexer near the adder will select complement of 16'b1. Thus Q - 1 will be performed. Lastly the result will be selected by MUX of which selection input is FunSel[1](0) and loaded into flip flops.
  - 01 : Increment the Q value of register. Since FunSel[0] is 1, Cin is 0 and multiplexer near the adder will select 16'b1. Thus Q + 1 will be performed. Lastly the result will be selected by MUX of which selection input is FunSel[1](0).

  - 10 : Load I into the register. Since one multiplexer's selection bit is 0 (FunSel[0]) and the other's is 1 FunSel[1], I input will be selected and loaded into flip flops.
  - 11 : Clear the register. Since one multiplexer's selection bit is 1 (FunSel[0]) and the other's is 1 FunSel[1], 16'b0 will be selected and loaded into flip flops.

- E - connected to all flip flops
  - 0 : Register is disabled
  - 1 : Register is enabled



Implementation of 8 bit register

For 16 bit register we used 16 1 bit D flip flop, 3 16 bit 2:1 Multiplexer and a 16 bit adder/substractor.

Part 2a

Inputs and Outputs:

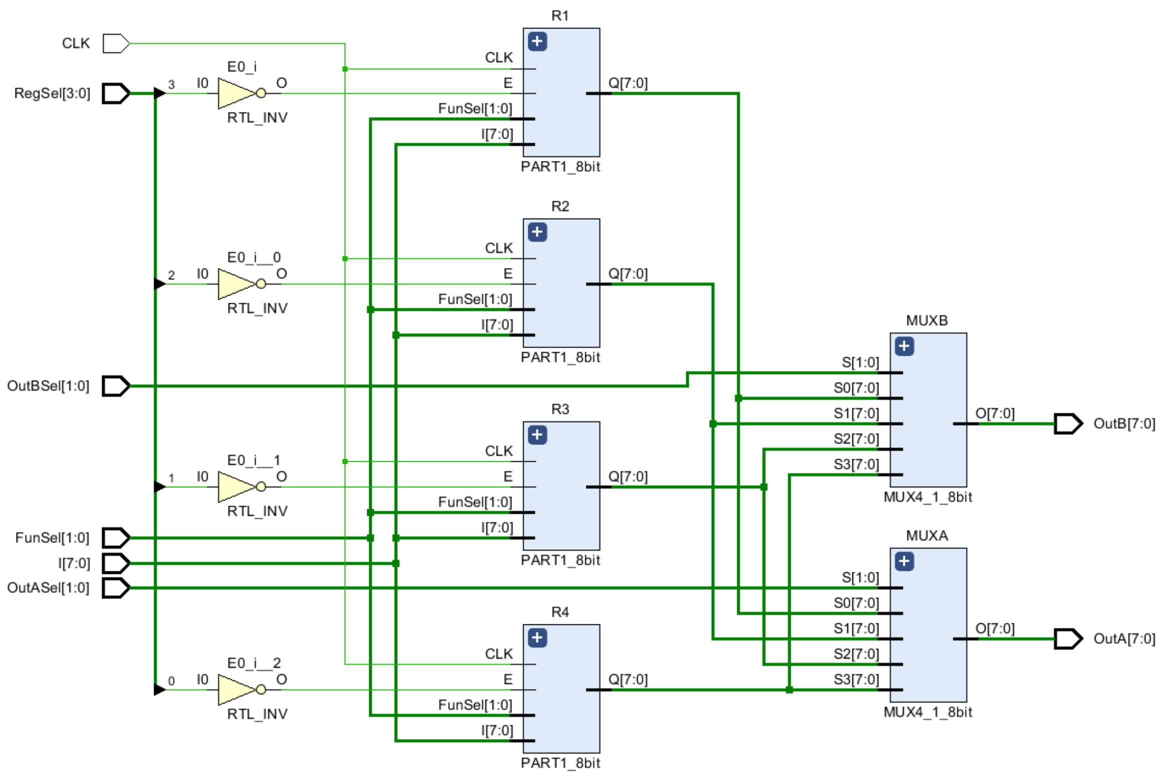
- Inputs:
  - OutASel and OutBSel inputs 2 bit control input that decide which register's output will be given as the output of system.
  - FunSel is a 2 bit input that decides which operation performs in the register.
  - RegSel is a 4 bit input that determines enabled registers.
  - I is the 8 bit input number that will be loaded to the registers.
- Outputs:

  - OutA and OutB are 8 bit outputs. They give the output of register determined by OutASel and OutBSel.

While implementing the circuit, we gave complements of RegSel's each bit to enable input of registers (RegSel[0] to R4, RegSel[1] to R3, RegSel[2] to R2 and RegSel[3] to R1). FunSel and I input are directly are given to every register. Clock is connected to every register, thus they work synchronically. Lastly, outputs are selected with multiplexer one of which is for OutA, and the other for OutB.

Control inputs

- OutASel and OutBSel - connected to separate 4:1 Multiplexers
  - 00 : R1 will be selected as output of multiplexers
  - 01 : R2 will be selected as output of multiplexers
  - 10 : R3 will be selected as output of multiplexers
  - 11 : R4 will be selected as output of multiplexers
- FunSel is directly sent to registers as FunSel inputs
  - 00 : Decrement register's value
  - 01 : Increment register's value
  - 10 : Load I into registers
  - 11 : Clear registers
- RegSel - RegSel[0] connected to R4, RegSel[1] connected to R3, RegSel[2] connected to R2, RegSel[3] connected to R1.
  - RegSel[0] = 0 : R4 is enabled
  - RegSel[1] = 0 : R3 is enabled
  - RegSel[2] = 0 : R2 is enabled
  - RegSel[3] = 0 : R1 is enabled



Implementation of Part2a

Part 2b

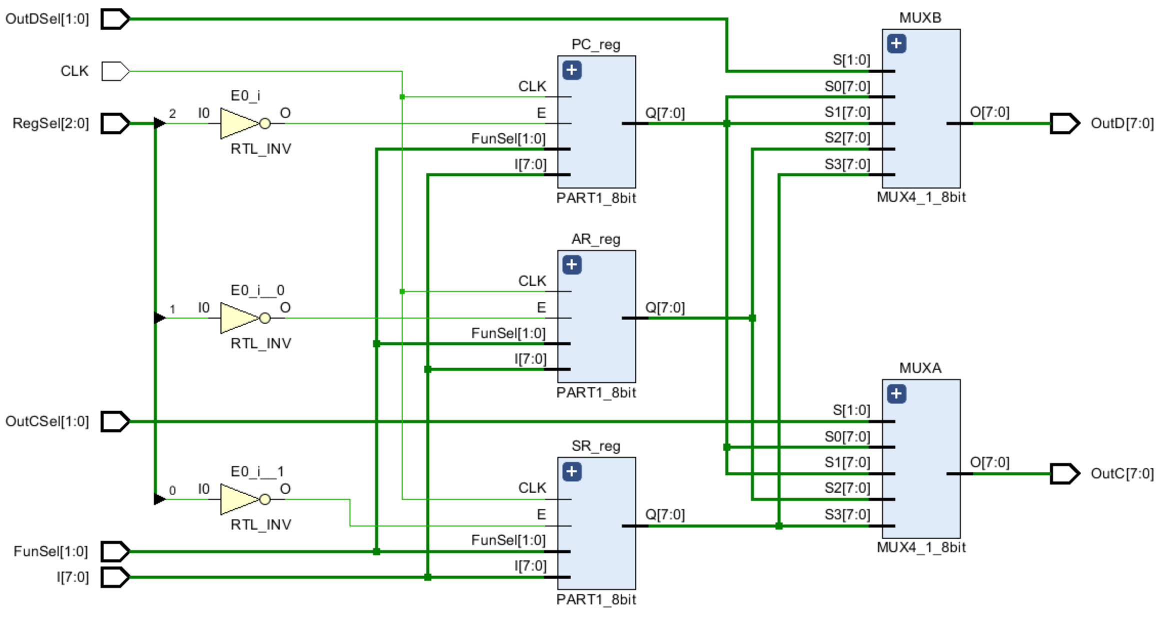
Inputs and Outputs:

- Inputs:
  - OutCSel and OutDSel inputs 2 bit control input that decide which register's output will be given as the output of system.
  - FunSel is a 2 bit input that decides which operation performs in the register.
  - RegSel is a 3 bit input that determines enabled registers.
  - I is the 8 bit input number that will be loaded to the registers.
- Outputs:
  - OutC and OutD are 8 bit outputs. They give the output of register determined by OutCSel and OutDSel.

We implemented this circuit in the same way as part2a. Only difference is that there is one less register in the circuit and RegSel is 3 bit. 2 input of multiplexers give same output (Selection 00 or 01 output is PC)

Control inputs

- OutCSel and OutDSel - connected to separate 4:1 Multiplexers
  - 00 : PC will be selected as output of multiplexers
  - 01 : PC will be selected as output of multiplexers
  - 10 : AR will be selected as output of multiplexers
  - 11 : SP will be selected as output of multiplexers
- FunSel is directly sent to registers as FunSel inputs
  - 00 : Decrement register's value
  - 01 : Increment register's value
  - 10 : Load I into registers
  - 11 : Clear registers
- RegSel - RegSel[0] connected to PC, RegSel[1] connected to AR, RegSel[2] connected to SR.
  - RegSel[0] = 0 : PC is enabled
  - RegSel[1] = 0 : AR is enabled
  - RegSel[2] = 0 : SR is enabled



Implementation of Part2b

Part 2c

Inputs and Outputs:

- Inputs:
  - E is enable input, when it is 0, output of the register will not change no matter what you give as input.
  - FunSel is a 2 bit input that decides which operation performs in the register. 00 decrement, 01 increment, 10 load and 11 clear.
  - I is the 8 bit input number that will be loaded to the registers.
  - L'/H is 1 bit input that selects the position of IR register's bits where I will be loaded. If it is 1, I will be loaded into IR(7-0), otherwise it will be loaded into IR(15-8).
- Outputs:
  - IRout is the 16 bit output of the register used in this part.

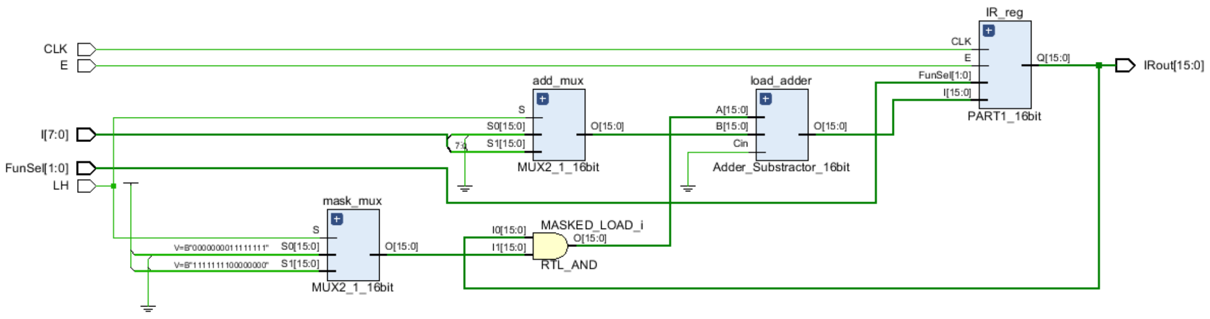
In this part, we used 16 bit register created in part1. In order to load I into bit positions of register determined by L'H, we used mask and add operations. For instance, if the value register have is 0100100001010101 and we need to load 11111111 into IR(15-8). We perform a mask operation 0100100001010101 . 0000000001010101 and clear the positions where we will load I. Then we add I into a temp variable's first 8 bits from left side and fill the other bits with 0 to match the lengths of I and IR. Lastly we add them (0000000001010101 + 1111111100000000) with a 16 bit adder and load the result into register.(1111111101010101).

Control inputs

- L'.H - connected to two 2:1 multiplexers
  - 0 : If FunSel is 10, load I into IR(15-8). From mask mux, it selects 0000000011111111 to mask first 8 bits (IR(15-8)) of register's current value. From add mux, it selects I input that is extended to 16 bit by adding zeros to right side. Then these two values will be added and the result will be loaded to register.
  - 1 : If FunSel is 10, load I into IR(7-0). From mask mux, it selects 1111111100000000 to mask first 8 bits (IR(7-0)) of register's current value. From add mux, it selects I input that is extended to 16 bit by adding zeros to

left side. Then these two values will be added and the result will be loaded to register.

- FunSel
  - 00 : Decrement register's value
  - 01 : Increment register's value
  - 10 : Load I into IR(15-8) if LH is 0, otherwise into IR(7-0)
  - 11 : Clear register
- E
  - 0 : Register is enabled
  - 1 : Register is disabled



Implementation of Part2c

Part 3

Inputs and Outputs:

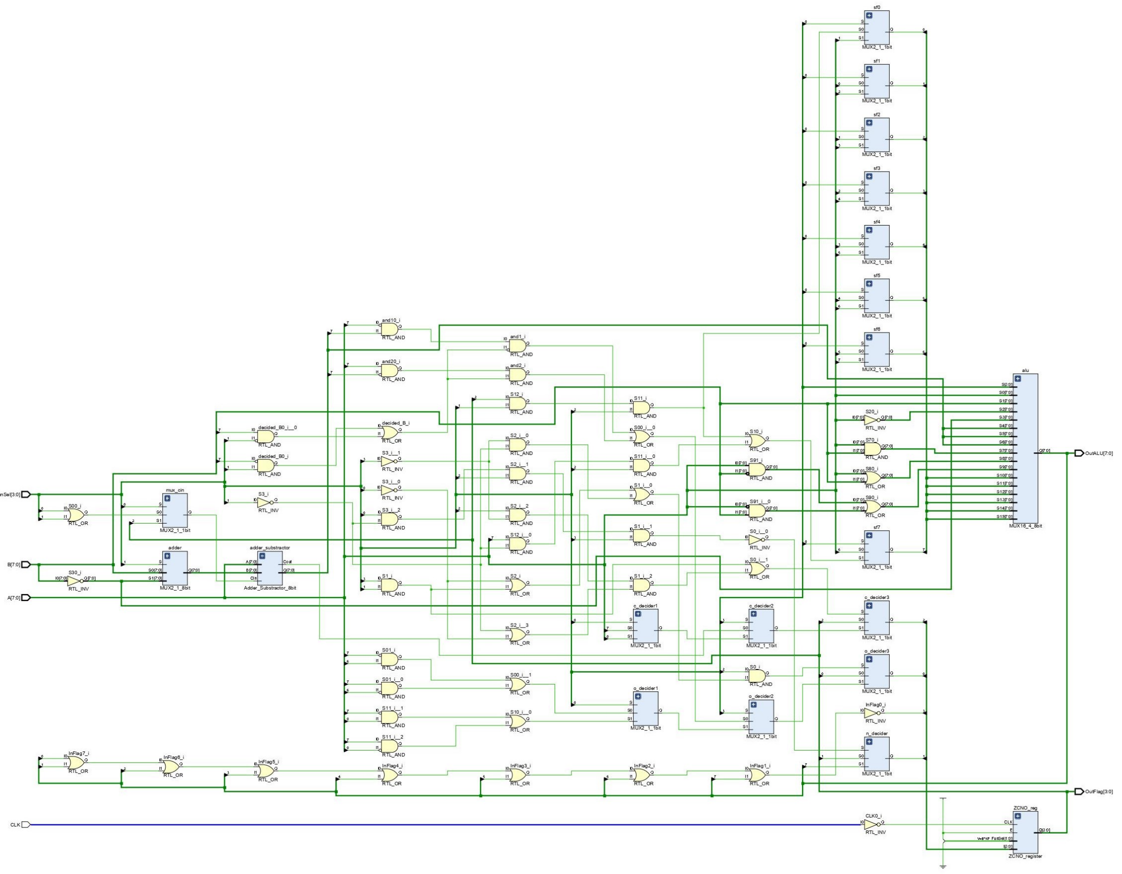
- Inputs:
  - FunSel is a 4 bit input that decides which operation performs in the ALU.
  - A and B are 8 bit inputs which operands for the ALU's operation.
- Outputs:
  - OutALU is the 8 bit output of the ALU.
  - Z, C, N and O are 1 bit flags in ZCNO register. Z is set when OutALU is zero, C is set when the operation performed inside ALU has a carry, N is set when OutALU is a negative number and O flag is set when overflow occurs in the operation performed inside ALU.

In this part, we design Arithmetic Logic Unit (ALU) that has two 8-bit inputs and an 8-bit output. We used inverter for complement operations; full-adders (which is created by us) for add and subtract operations; AND and OR gates for logical operations; a system that we build with multiplexers (selection input of multiplexers are FunSel[0]) for shift operations. We wrote 4-bit register module for OutFlag register.

Control inputs

- FunSel
  - 0000 - 0001 : When FunSel[0] is 0, OutALU will be equal to A.- When FunSel[0] is 1, OutALU will be equal to B. Also, only Z and N bits of OutFlag register can be change by ALU.
  - 0010 - 0011 : When FunSel[0] is 0, OutALU will be equal to Complement of A. When FunSel[0] is 1, OutALU will be equal to Complement of B. Also, only Z and N bits of OutFlag register can be change by ALU.
  - 0100 : A and B values are added by Half Adder(when Cin = 0, Full Adder) and outALU will be equal to result of A+B. Also, all bits of OutFlag register can be change by ALU.
  - 0101 : A and B values are added by Full Adder, and OutALU will be equal to result of A+B+Carry. Also, all bits of OutFlag register can be change by ALU.
  - 0110 : A and Complement of B values are added by Full Adder when Cin = 1. OutALU will be equal to result of A-B. Also, all bits of OutFlag register can be change by ALU.
  - 0111 : OutALU will be equal to the result of the BITWISE AND operation with inputs A and B. Also, Also, only Z and N bits of OutFlag register can be change by ALU.
  - 1000 : OutALU will be equal to the result of the BITWISE OR operation with inputs A and B. Also, Also, only Z and N bits of OutFlag register can be change by ALU.
  - 1001 : OutALU will be equal to the result of the BITWISE XOR operation with inputs A and B. Also, Also, only Z and N bits of OutFlag register can be change by ALU.
  - 1010 : OutALU[0] will be equal to 0, and OutALU[i+1] is equal to A[i] when i is between 0 and 6. Also, C bit of OutFlag register is equal to A[7]. Only Z, C and N bits of OutFlag register can be change by ALU.

- 1011 : OutALU[7] will be equal to 0, and OutALU[i] is equal to A[i+1] when i is between 0 and 6. Also, C bit of OutFlag register is equal to A[0]. Only Z, C and N bits of OutFlag register can be change by ALU.
- 1100 : OutALU[0] will be equal to 0, and OutALU[i+1] is equal to A[i] when i is between 0 and 6. Only Z, N and O bits of OutFlag register can be change by ALU. When (Outflag[7] XOR A[7]) is equal to 1, Overflow Flag is equal to 1.
- 1101 : OutALU[7] will be equal to A[7], and OutALU[i] is equal to A[i+1] when i is between 0 and 6. Only Z bit of OutFlag register can be change by ALU.
- 1110 : OutALU[0] will be equal to value of Carry flag, and OutALU[i+1] is equal to A[i] when i is between 0 and 6. Also, all bits of OutFlag register can be change by ALU. When (Outflag[7] XOR A[7]) is equal to 1, Overflow Flag is equal to 1. C bit of OutFlag register is equal to A[7].
- 1111 :OutALU[7] will be equal to value of Carry flag, and OutALU[i] is equal to A[i+1] when i is between 0 and 6. Also, all bits of OutFlag register can be change by ALU. When (Outflag[7] XOR A[7]) is equal to 1, Overflow Flag is equal to 1. C bit of OutFlag register is equal to A[0].



Implementation of Part3

Part 4

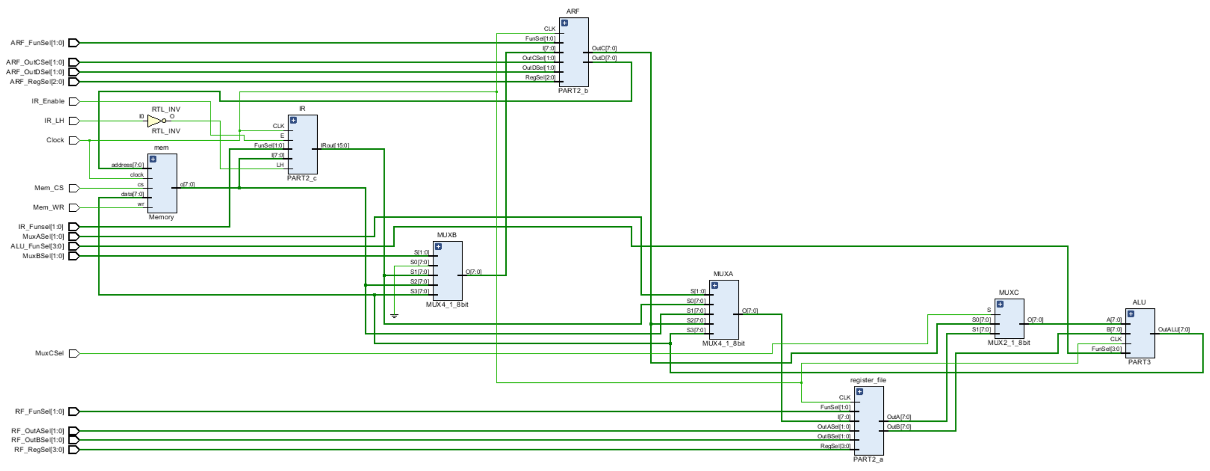
Inputs and Outputs:

- Inputs:
  - RF OutASel and RF OutBSel inputs 2 bit control input that decide which register's output will be given as the output of system.
  - RF FunSel is a 2 bit input that decides which operation performs in the register
  - RF RegSel is a 4 bit input that determines enabled registers
  - ALU FunSel is a 4 bit input that decides which operation performs in the ALU.
  - ARF OutCSel ARF OutDSel inputs 2 bit control input that decide which register's output will be given as the output of system.
  - ARF FunSel is a 2 bit input that decides which operation performs in the register.
  - ARF RegSel is a 3 bit input that determines enabled registers.
  - IR LH is 1 bit input that selects the position of IR register's bits where I will be loaded. If it is 1, I will be loaded into IR(7-0), otherwise it will be loaded into IR(15-8).
  - IR Enable is enable input, when it is 0, output of the register will not change no matter what you give as input.
  - IR Funsel is a 2 bit input that decides which operation performs in the register. 00 decrement, 01 increment, 10 load and 11 clear
  - Mem WR is control input for write or read. While WR is equal to 0, Chip is in read mode and while WR is equal to 1, Chip is in write mode.
  - Mem CS is enable input, when it is 0, Chip is enable.
  - MuxASel is a 2 bit input that selects inputs of MUX A.
  - MuxBSel is a 2 bit input that selects inputs of MUX B.
  - MuxCSel is a 1 bit input that selects inputs of MUX C.

In this part, we created a system with memory and modules which we built in previous parts (IR register, RF, ARF and ALU). We used 3 multiplexers which called MuxASel, MuxBSel and MuxCSel to organize the connections.

Control inputs

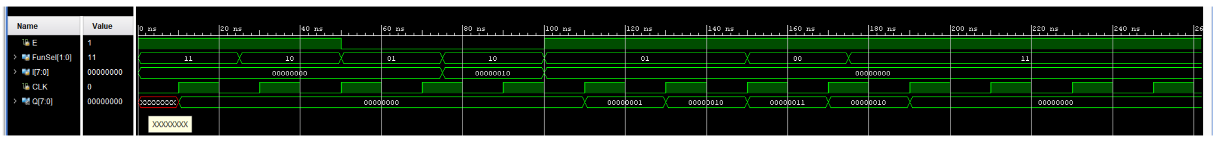
- MuxASel
  - 00 : Mux A sends sends first 8 bits of Output of IR to input of Register File
  - 01 : Mux A sends memory output to input of Register File.
  - 10 : Mux A sends C output of Address Register File to input of Register File.
  - 11 : Mux A sends output of ALU to input of Register File.
- MuxBSel
  - 00 : Don't-Care
  - 01 : Mux B sends first 8 bits of Output of IR to input of Address Register File.
  - 10 : Mux B sends memory output to input of Address Register File.
  - 11 : Mux B sends output of ALU to input of Address Register File.
- MuxCSel
  - 0 : Mux C sends ARF to A input of ALU.
  - 1 : Mux C sends A output of register file to A input of ALU.



Implementation of Part4

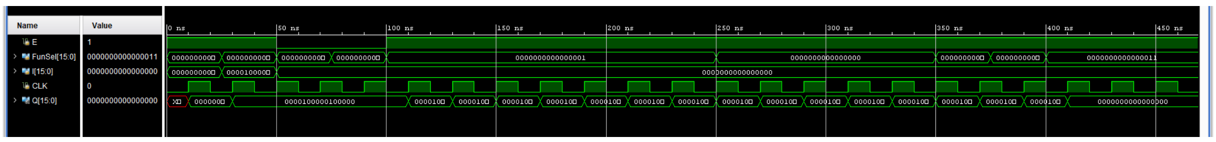
3 TEST CASES AND RESULTS

Part 1 (8 bits)



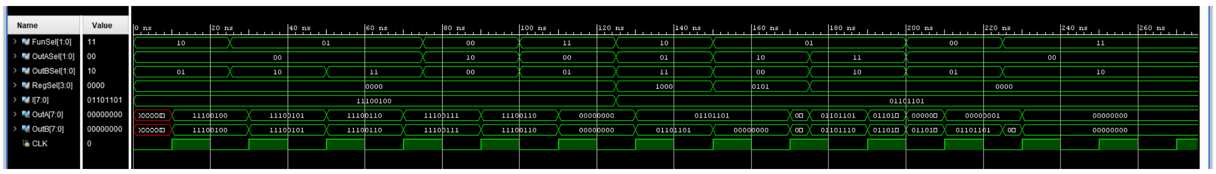
The results of 8 bit register simulation

Part 1 (16 bits)



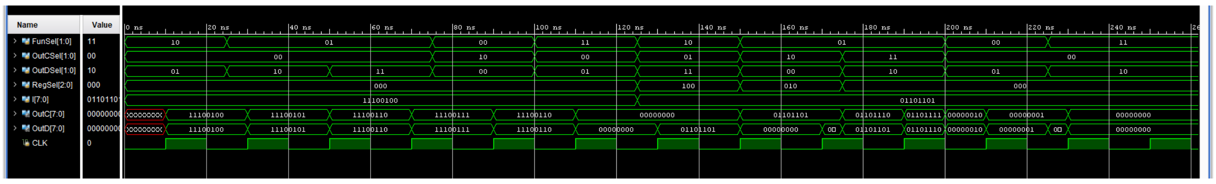
The results of 16 bit register simulation

Part 2a



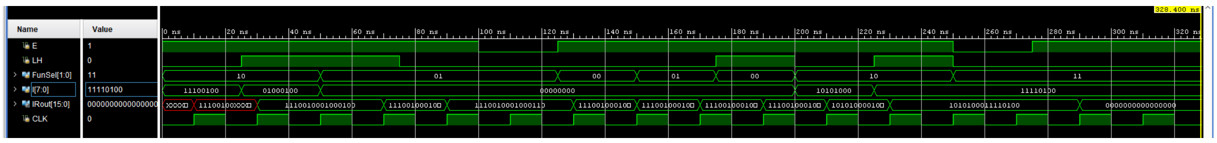
The results of 8 bit register file simulation

Part 2b



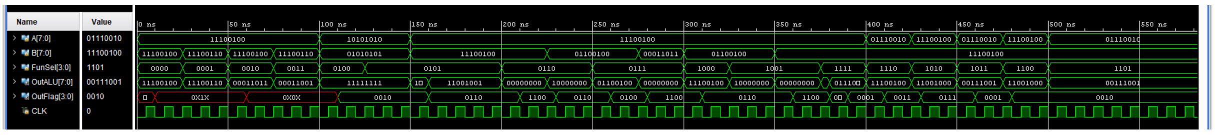
The results of 8 bit address register file simulation

Part 2c



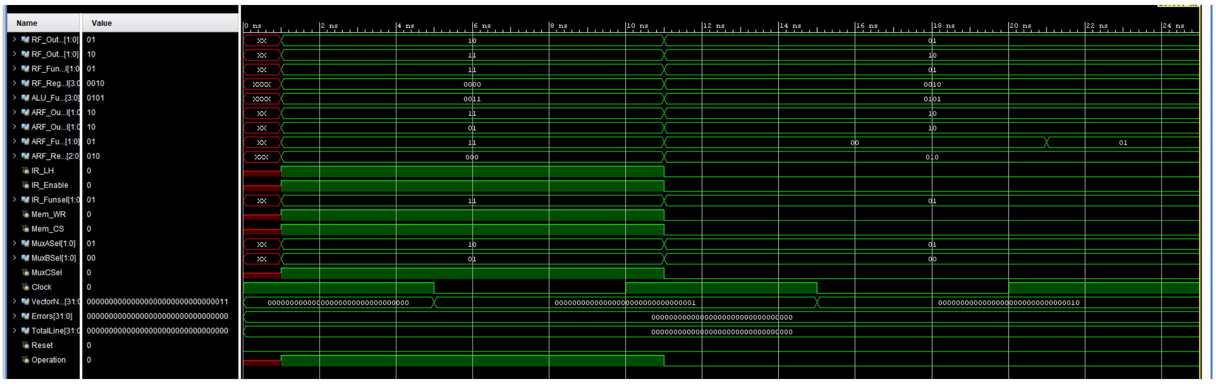
The results of 16 bit IR register simulation

Part 3

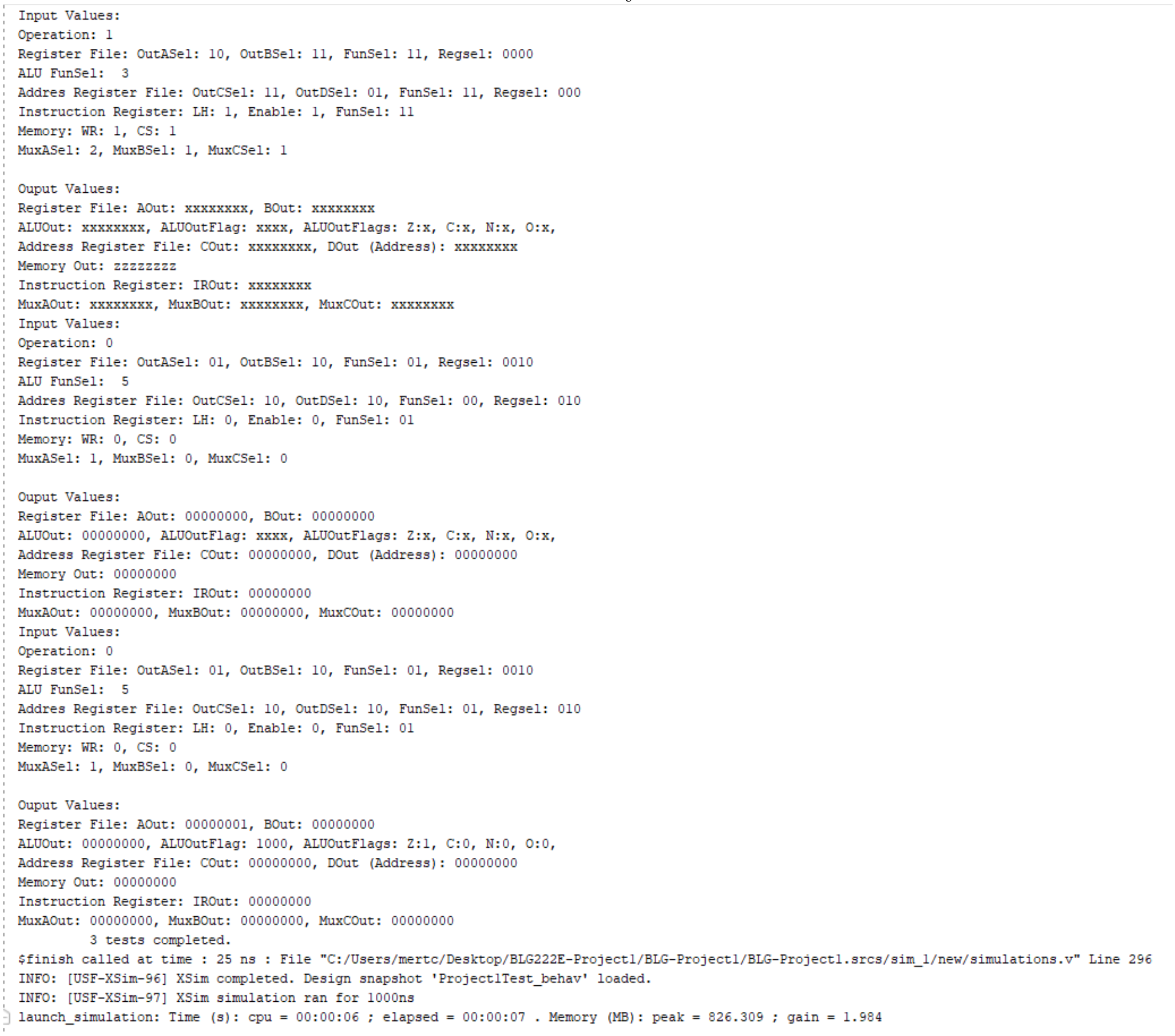


The results of 8 bit Arithmetic Logic Unit simulation

Part 4



The results of whole system simulation



```
Input Values:
Operation: 1
Register File: OutASel: 10, OutBSel: 11, FunSel: 11, Regsel: 0000
ALU FunSel:  3
Addres Register File: OutCSel: 11, OutDSel: 01, FunSel: 11, Regsel: 000
Instruction Register: LH: 1, Enable: 1, FunSel: 11
Memory: WR: 1, CS: 1
MuxASel: 2, MuxBSel: 1, MuxCSel: 1

Ouput Values:
Register File: AOut: xxxxxxxx, BOut: xxxxxxxx
ALUOut: xxxxxxxx, ALUOutFlag: xxxx, ALUOutFlags: Z:x, C:x, N:x, O:x,
Address Register File: COut: xxxxxxxx, DOut (Address): xxxxxxxx
Memory Out: zzzzzzzz
Instruction Register: IROut: xxxxxxxx
MuxAOut: xxxxxxxx, MuxBOut: xxxxxxxx, MuxCOut: xxxxxxxx
Input Values:
Operation: 0
Register File: OutASel: 01, OutBSel: 10, FunSel: 01, Regsel: 0010
ALU FunSel:  5
Addres Register File: OutCSel: 10, OutDSel: 10, FunSel: 00, Regsel: 010
Instruction Register: LH: 0, Enable: 0, FunSel: 01
Memory: WR: 0, CS: 0
MuxASel: 1, MuxBSel: 0, MuxCSel: 0

Ouput Values:
Register File: AOut: 00000000, BOut: 00000000
ALUOut: 00000000, ALUOutFlag: xxxx, ALUOutFlags: Z:x, C:x, N:x, O:x,
Address Register File: COut: 00000000, DOut (Address): 00000000
Memory Out: 00000000
Instruction Register: IROut: 00000000
MuxAOut: 00000000, MuxBOut: 00000000, MuxCOut: 00000000
Input Values:
Operation: 0
Register File: OutASel: 01, OutBSel: 10, FunSel: 01, Regsel: 0010
ALU FunSel:  5
Addres Register File: OutCSel: 10, OutDSel: 10, FunSel: 01, Regsel: 010
Instruction Register: LH: 0, Enable: 0, FunSel: 01
Memory: WR: 0, CS: 0
MuxASel: 1, MuxBSel: 0, MuxCSel: 0

Ouput Values:
Register File: AOut: 00000001, BOut: 00000000
ALUOut: 00000000, ALUOutFlag: 1000, ALUOutFlags: Z:1, C:0, N:0, O:0,
Address Register File: COut: 00000000, DOut (Address): 00000000
Memory Out: 00000000
Instruction Register: IROut: 00000000
MuxAOut: 00000000, MuxBOut: 00000000, MuxCOut: 00000000
           3 tests completed.
$finish called at time : 25 ns : File "C:/Users/mertc/Desktop/BLG222E-Project1/BLG-Project1/BLG-Project1.srcs/sim_1/new/simulations.v" Line 296
INFO: [USF-XSim-96] XSim completed. Design snapshot 'Project1Test_behav' loaded.
INFO: [USF-XSim-97] XSim simulation ran for 1000ns
launch_simulation: Time (s): cpu = 00:00:06 ; elapsed = 00:00:07 . Memory (MB): peak = 826.309 ; gain = 1.984
```

4 DISCUSSION

In this project, first we implemented 8 bit and 16 bit registers, which have load, clear, E and FunSel inputs, and these registers performed different operation according to FunSel input.

In part 2-a, we created Register file by using 4 8-bit registers which we implement in previous part. By using FunSel and RegSel inputs, we make some operations on these registers such as increment, decrement etc. We used 2 multiplexers to select output registers based on the OutASel and OutBSel control inputs. In part 2-b, we designed an address register file by using 3 8-bit registers which we implement in part 1. We used 2 multiplexers to select output registers based on the OutCSel and OutDSel control inputs. By using FunSel and RegSel inputs, we make some operations on these address registers such as increment, decrement etc. In part 2-c, we designed a 16-bit IR register by using a 16-bit register which we implement in part 1. When Enable input is 1, we make some operations on these address registers such as increment, decrement, load and clear according to FunSel input. In order to load I into bit positions of register determined by L'H, we used mask and add operations.

In part 3, we design Arithmetic Logic Unit (ALU) that has two 8-bit inputs and an 8-bit output. We wrote 4-bit register module for OutFlag register. We used full-adders (which is created by us) for add and subtract operations. To take the complements of A and B, we designed a circuit formed by inverters. To make the logical operations, we implement bitwise AND and OR gates. We created a multiplexer system which shifts to the left when FunSel[0] is equal to 0, and to the right when FunSel[0] is equal to 1. We select the output of these operations by using 16:1 MUX(selection inputs of multiplexer are FunSel) which we wrote as a module.

In the last part of the project, we designed a basic computer organization by combining modules which we created on previous parts. The whole system uses the same clock signal.

5 CONCLUSION

In this project we created a computer organization consist of multiplexers, memory, ALU and different registers. With the help of verilog and our knowledge from BLG 231E and BLG222E, we created different materials in each part and combined them to create our system.