ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 242E
DIGITAL CIRCUITS LABORATORY EXPERIMENT REPORT

PROJECT NO	:	2
DUE DATE	:	02.06.2022
GROUP NO	:	G62

GROUP MEMBERS:

150210710 : Fatih Baskın

SPRING 2022

Contents

1 INTRODUCTION

In this project, we designed the control circuitry for ALU System that we have designed in the first project. We used case structure and if-else structure quite extensively.

Also, a test case was provided for the ALU System. We examined the test case, diagnosed the problems and solved them.

And finally, we created quite extensive simulations to examine the control circuit.

2 PROPOSED INSTRUCTIONS AND DESIGN

2.1 Test Case for ALU System

For the ALU System, there were 11 test cases, our contraption did fail in the last two test cases. Here is the relevant parts of the TCL Console output:

```
Ouput Values:
Memory Out: 10101010
##############################################
  9 Line Check Started To Test
Line passed the test
##############################################
Input Values:
Instruction Register: LH: 0, Enable: 1, FunSel: 2
Ouput Values:
Memory Out: 01010101
Instruction Register: IROut:   0
##############################################
 10 Line Check Started To Test
IROut Error: Expected:   170, Actual:   0
LineError:   1/  3, Total Error:   1/ 10
##############################################
Input Values:
Instruction Register: LH: 0, Enable: 0, FunSel: 3
Ouput Values:
Instruction Register: IROut:  85
##############################################
```

```
  11 Line Check Started To Test
IROut Error: Expected: 21930, Actual:   85
LineError:    1/   2, Total Error:    2/ 11
##############################################
        13 tests completed with    2 errors.
```

The error was resulting from wrong instruction register design. LH signal loads low and high order bits incorrectly, also in ALU System, we accidently connected 8-bit wire to output of instruction register. We solved those issues and test completed without errors. We negated the LH wire for inputs of IR and also expanded that problematic cable to 16-bits. Also for observations in final system, we converted ALUOut, ALUOutFlag and IROut cables into outputs. Also we gave the outputs of the individual registers as outputs.

2.2 Instructions

Instruction register is 16-bits long but memory connection is 8-bit. Therefore instruction is fetched in two clock cycles. In first clock cycle, LSB should be loaded then the MSB should be loaded.

- T_0 : $IR(7-0) \leftarrow M[PC], PC \leftarrow PC + 1$ Loads LSB into IR
- T_1 : $IR(15-0) \leftarrow M[PC], PC \leftarrow PC + 1$ Loads MSB into IR

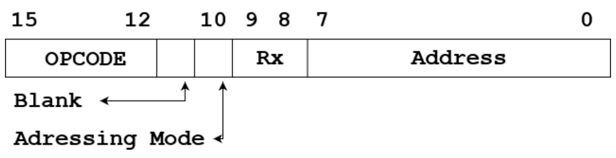


Figure 1: Instructions with an address reference

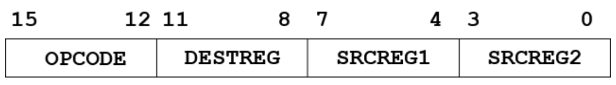
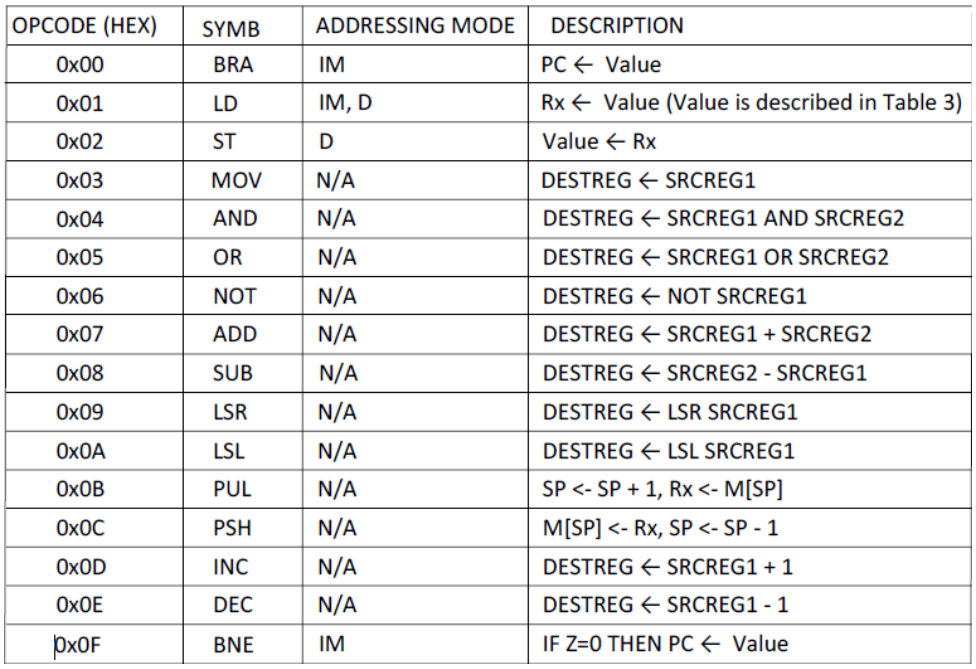


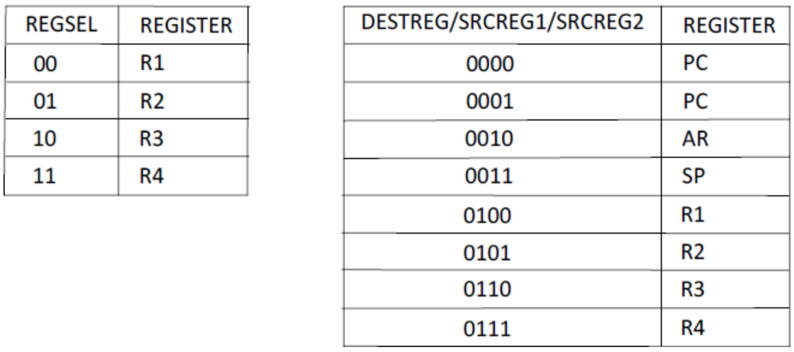
Figure 2: Instructions without an address reference

There are 16 instructions, as OPCODE is 4-bits long. Also with different register targets and sources, number of possible instructions grows significantly.



OPCODE (HEX)	SYMB	ADDRESSING MODE	DESCRIPTION
0x00	BRA	IM	PC ← Value
0x01	LD	IM, D	Rx ← Value (Value is described in Table 3)
0x02	ST	D	Value ← Rx
0x03	MOV	N/A	DESTREG ← SRCREG1
0x04	AND	N/A	DESTREG ← SRCREG1 AND SRCREG2
0x05	OR	N/A	DESTREG ← SRCREG1 OR SRCREG2
0x06	NOT	N/A	DESTREG ← NOT SRCREG1
0x07	ADD	N/A	DESTREG ← SRCREG1 + SRCREG2
0x08	SUB	N/A	DESTREG ← SRCREG2 - SRCREG1
0x09	LSR	N/A	DESTREG ← LSR SRCREG1
0x0A	LSL	N/A	DESTREG ← LSL SRCREG1
0x0B	PUL	N/A	SP <- SP + 1, Rx <- M[SP]
0x0C	PSH	N/A	M[SP] <- Rx, SP <- SP - 1
0x0D	INC	N/A	DESTREG ← SRCREG1 + 1
0x0E	DEC	N/A	DESTREG ← SRCREG1 - 1
0x0F	BNE	IM	IF Z=0 THEN PC ← Value

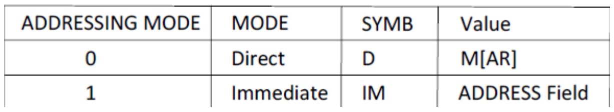
Figure 3: OPCODES and their explanations[1]



REGSEL	REGISTER
00	R1
01	R2
10	R3
11	R4

DESTREG/SRCREG1/SRCREG2	REGISTER
0000	PC
0001	PC
0010	AR
0011	SP
0100	R1
0101	R2
0110	R3
0111	R4

Figure 4: Register codes and their meanings inside the instruction[1]



ADDRESSING MODE	MODE	SYMB	Value
0	Direct	D	M[AR]
1	Immediate	IM	ADDRESS Field

Figure 5: Addressing modes and their explanation[1]

For SRCREG2, only the cases for register file(0x4, 0x5, 0x6, 0x7) are available.

2.3 Internal Design

We used Verilog's if-else and case structures to make our control unit. Therefore its design is auto-generated. However determining the correct outputs (inputs for ALU System) was quite challenging. We drove those inputs with cases. Other than that, our control unit has a 4-bit counter with reset switch, a system reset switch which writes zero to all registers, including ALU System and decoders for DESTREG and Rx. Also for some operations it takes ZCNO flags as inputs.

In each clock cycle our control unit changes several output signals (inputs of the clock signal). Here is the list:

RF_RegSel	ARF_RegSel	IR_Enable	MuxASel	ALU_FunSel
RF_FunSel	ARF_FunSel	IR_Funsel	MuxBSel	Mem_WR
RF_OutASel	ARF_OutCSel	IR_LH	MuxCSel	Mem_CS
RF_OutBSel	ARF_OutDSel			reset_counter

Table 1: Control Unit's control outputs

3 EXTENSIVE TESTING AND SIMULATIONS

3.1 Fetch

- $T_0 : IR(7-0) \leftarrow M[PC], PC \leftarrow PC + 1$ Loads LSB into IR
- $T_1 : IR(15-0) \leftarrow M[PC], PC \leftarrow PC + 1$ Loads MSB into IR

In first two clock cycles (T_0 and T_1) we fetch the instructions LSB and MSB respectively. Therefore control unit's outputs should be like:

RF_RegSel	1111	ARF_RegSel	011	IR_Enable	1	MuxASel	Φ	ALU_FunSel	Φ
RF_FunSel	Φ	ARF_FunSel	01	IR_Funsel	10	MuxBSel	Φ	Mem_WR	0
RF_OutASel	Φ	ARF_OutCSel	Φ	IR_LH	1	MuxCSel	Φ	Mem_CS	0
RF_OutBSel	Φ	ARF_OutDSel	Φ					reset_counter	0

Table 2: Control Unit's control outputs in T_0

Don't modify RF, increment PC, write on IR's LSB (LH = 1), don't write on memory and open memory output.

RF_RegSel	1111	ARF_RegSel	011	IR_Enable	1	MuxASel	Φ	ALU_FunSel	Φ
RF_FunSel	Φ	ARF_FunSel	01	IR_Funsel	10	MuxBSel	Φ	Mem_WR	0
RF_OutASel	Φ	ARF_OutCSel	Φ	IR_LH	0	MuxCSel	Φ	Mem_CS	0
RF_OutBSel	Φ	ARF_OutDSel	00					reset_counter	0

Table 3: Control Unit's control outputs in T_1

Don't modify RF, increment PC, write on IR's MSB (LH = 0), don't write on memory and open memory output.

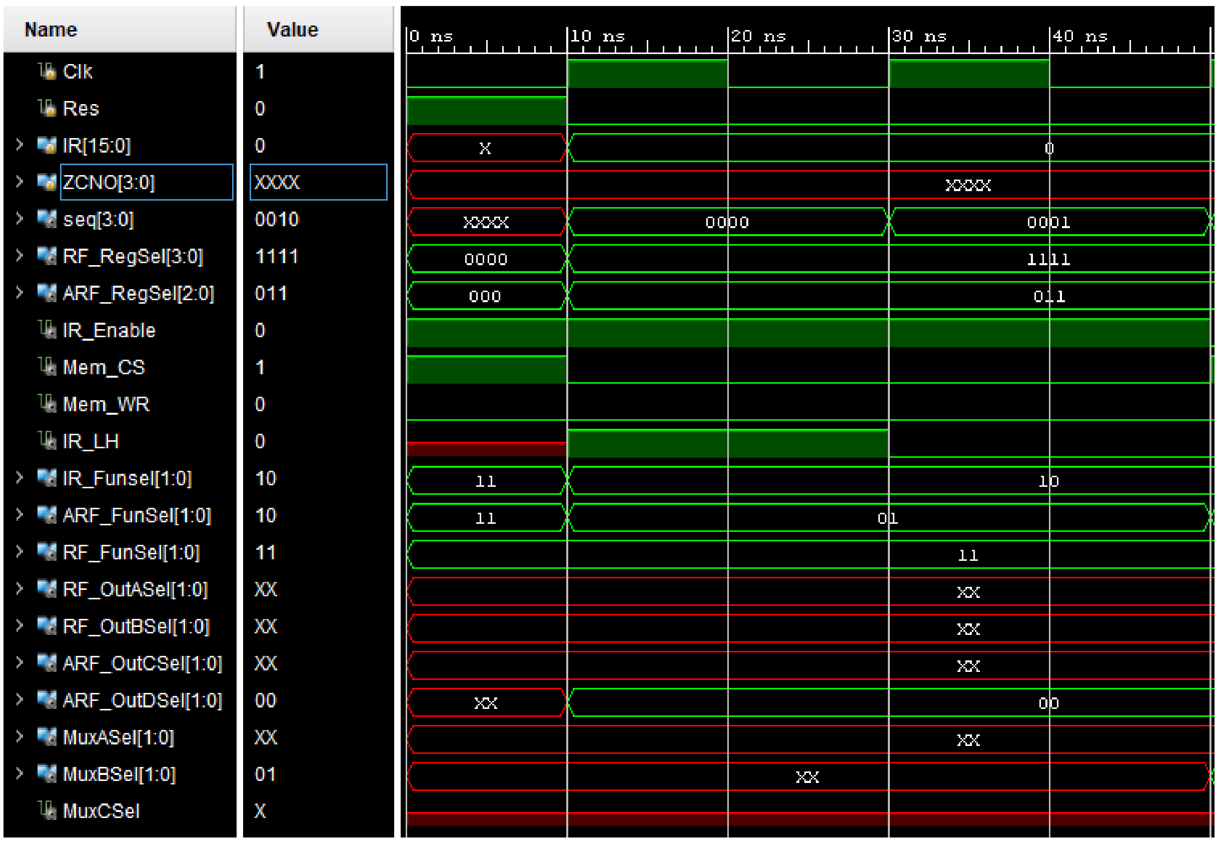


Figure 6: Simulation of fetch phase (seq: sequence counter)

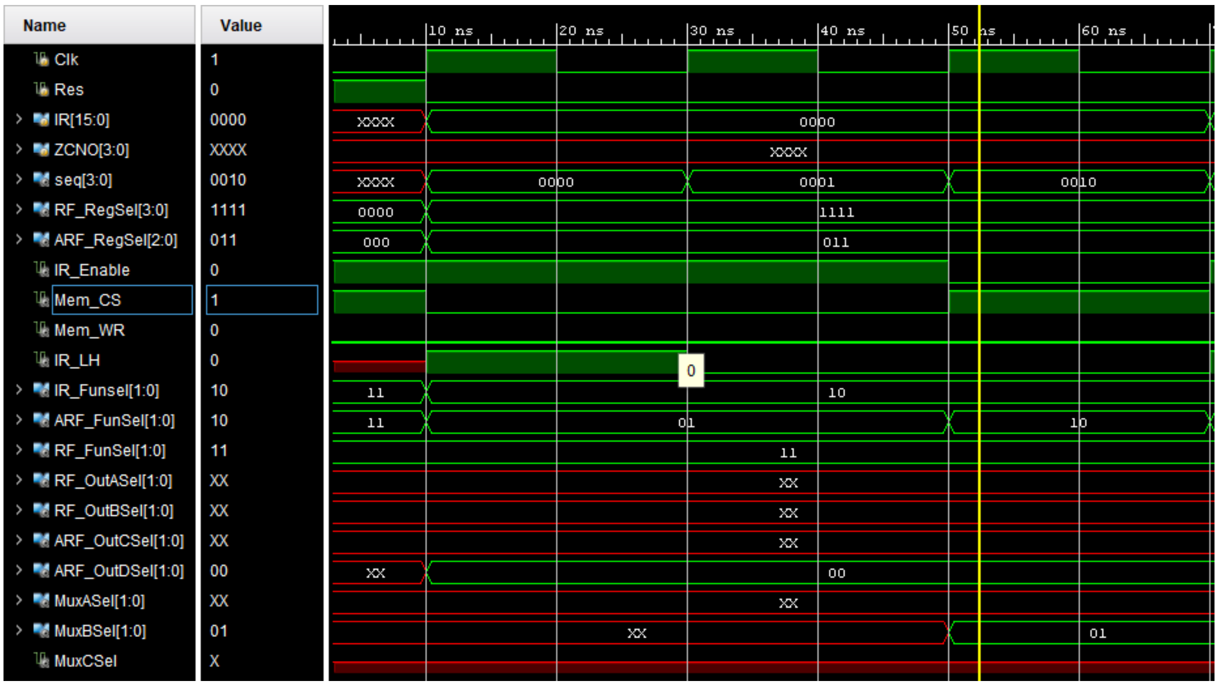
3.2 OPCODE 0x0

This instruction is for branching. It's symbol is BRA, addressing mode is immediate and RTL description is: $PC \leftarrow Value$. In this case value will be instruction register's address filed.

- T_2 : $PC \leftarrow IR(7-0)$, $SC \leftarrow 0$

RF_RegSel	1111	ARF_RegSel	011	IR_Enable	0	MuxASel	Φ	ALU_FunSel	Φ
RF_FunSel	Φ	ARF_FunSel	10	IR_Funsel	Φ	MuxBSel	01	Mem_WR	0
RF_OutASel	Φ	ARF_OutCSel	Φ	IR_LH	Φ	MuxCSel	Φ	Mem_CS	1
RF_OutBSel	Φ	ARF_OutDSel	Φ					reset_counter	1

Table 4: Control Unit's control outputs for OPCODE 0x0 in T_2



Name	Value
Clk	1
Res	0
IR[15:0]	0000
ZCNO[3:0]	XXXX
seq[3:0]	0010
RF_RegSel[3:0]	1111
ARF_RegSel[2:0]	011
IR_Enable	0
Mem_CS	1
Mem_WR	0
IR_LH	0
IR_Funsel[1:0]	10
ARF_FunSel[1:0]	10
RF_FunSel[1:0]	11
RF_OutASel[1:0]	XX
RF_OutBSel[1:0]	XX
ARF_OutCSel[1:0]	XX
ARF_OutDSel[1:0]	00
MuxASel[1:0]	XX
MuxBSel[1:0]	01
MuxCSel	X

Figure 7: Simulation of OPCODE: 0x0

3.3 OPCODE 0x1

This instruction is for loading. It's symbol is LD, addressing mode is both direct (IR(10) = 0) immediate (IR(10) = 1) and RTL description is: $Rx \leftarrow Value$. In this case value will be either M[AR] or instruction register's address filed.

- $T_2\overline{IR(10)}$: $Rx \leftarrow M[AR]$, $SC \leftarrow 0$
- $T_2IR(10)$: $Rx \leftarrow IR(7-0)$, $SC \leftarrow 0$

RF_RegSel	Rx	ARF_RegSel	111	IR_Enable	0	MuxASel	01	ALU_FunSel	Φ
RF_FunSel	10	ARF_FunSel	Φ	IR_Funsel	Φ	MuxBSel	Φ	Mem_WR	0
RF_OutASel	Φ	ARF_OutCSel	Φ	IR_LH	Φ	MuxCSel	Φ	Mem_CS	0
RF_OutBSel	Φ	ARF_OutDSel	Φ					reset_counter	1

Table 5: Control Unit's control outputs for OPCODE 0x1 in $T_2\overline{IR(10)}$

RF_RegSel	Rx	ARF_RegSel	111	IR_Enable	0	MuxASel	00	ALU_FunSel	Φ
RF_FunSel	10	ARF_FunSel	Φ	IR_Funsel	Φ	MuxBSel	Φ	Mem_WR	0
RF_OutASel	Φ	ARF_OutCSel	Φ	IR_LH	Φ	MuxCSel	Φ	Mem_CS	1
RF_OutBSel	Φ	ARF_OutDSel	10					reset_counter	1

Table 6: Control Unit's control outputs for OPCODE 0x1 in $T_2IR(10)$

Note that Rx is obtained by decoding IR(9-8) according to design specifications.

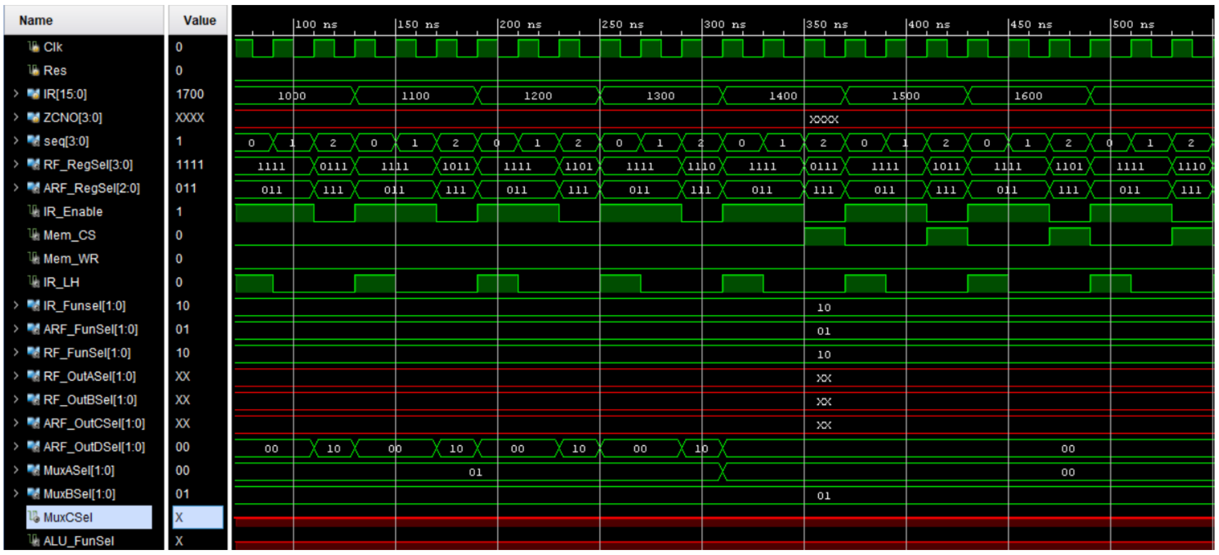


Figure 8: Simulation of OPCODE 0x1

3.4 OPCODE 0x2

This instruction is saving (storing) data into the memory. Symbol is ST. Adressing mode is direct. RTL description is $M[AR] \leftarrow Rx$.

- $T_2 : AR \leftarrow IR(7-0)$
- $T_3 : M[AR] \leftarrow Rx$

RF_RegSel	1111	ARF_RegSel	111	IR_Enable	0	MuxASel	Φ	ALU_FunSel	0x1
RF_FunSel	Φ	ARF_FunSel	Φ	IR_Funsel	Φ	MuxBSel	01	Mem_WR	0
RF_OutASel	Φ	ARF_OutCSel	Φ	IR_LH	Φ	MuxCSel	Φ	Mem_CS	1
RF_OutBSel	IR(9-8)	ARF_OutDSel	10					reset_counter	1

Table 7: Control Unit's control outputs for OPCODE 0x2 in T_2

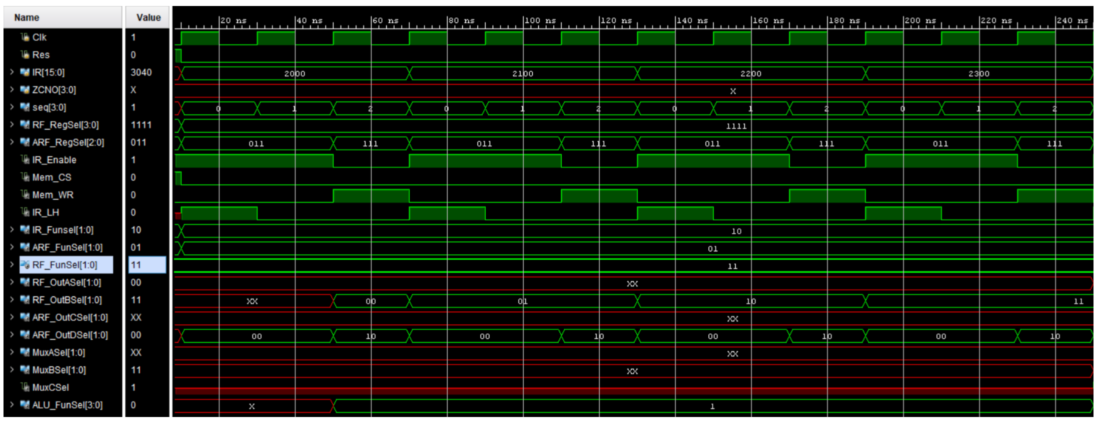


Figure 9: Simulation for OPCODE 0x2 with different Rx'es

3.5 OPCODE 0x3

This instruction is for transferring (moving) data between register. Its symbol is MOV. RTL description is $DESTREG \leftarrow SCRREG1$. This instruction is one clock cycle long.

RF_RegSel	Y	ARF_RegSel	Y	IR_Enable	0	MuxASel	11	ALU_FunSel	0x0
RF_FunSel	10	ARF_FunSel	10	IR_Funsel	Φ	MuxBSel	11	Mem_WR	0
RF_OutASel	X	ARF_OutCSel	X	IR_LH	Φ	MuxCSel	Z	Mem_CS	1
RF_OutBSel	Φ	ARF_OutDSel	Φ					reset_counter	1

Table 8: Control Unit's control outputs for OPCODE 0x3

- X : IR(5-4).
- Y : RegSel outputs are determined by DESTREG.
- Z : MuxCSel is determined by SRCREG, if SRCREG is inside RF then MuxCSel = 1, else MuxCSel = 0.

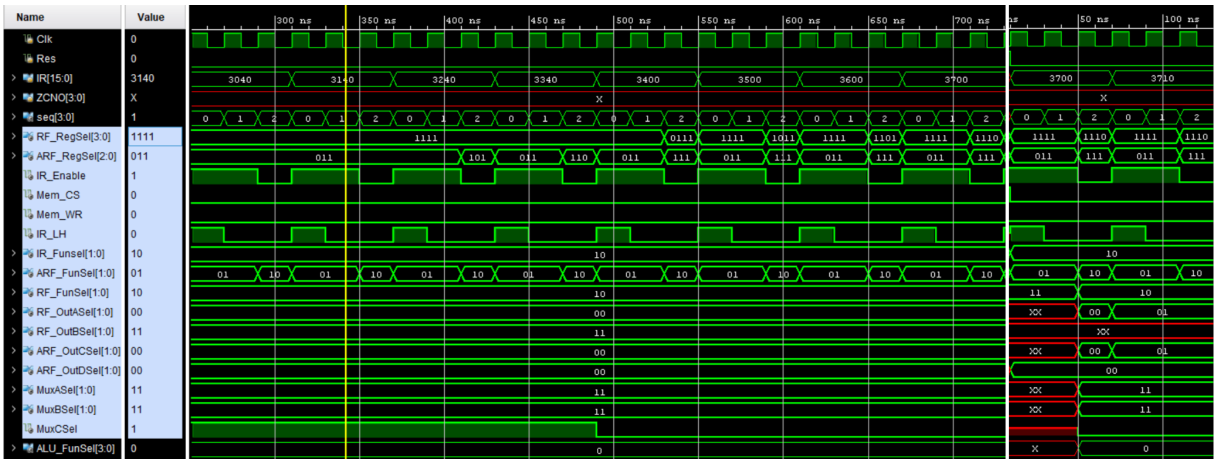


Figure 10: Simulation of OPCODE 0x3 part 1

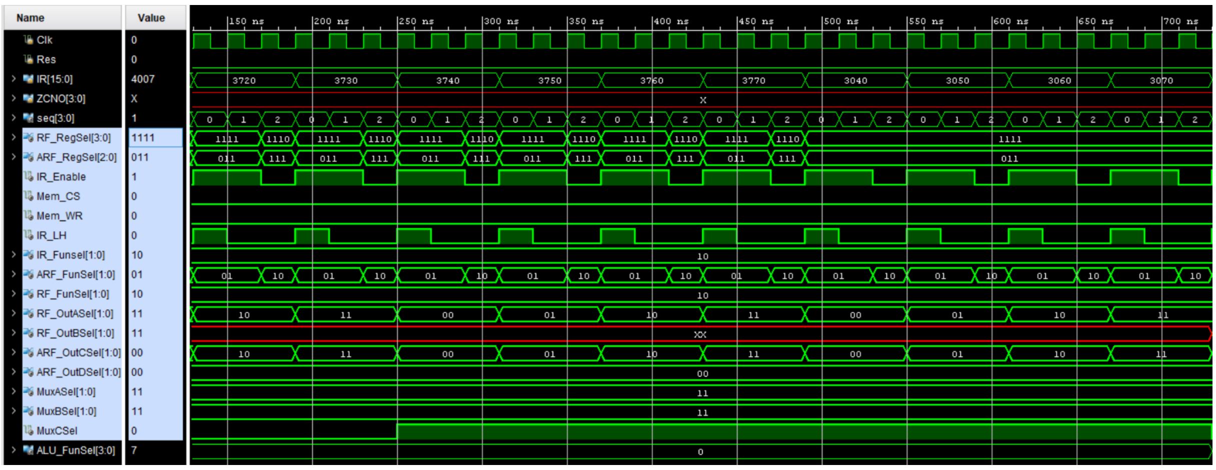


Figure 11: Simulation of OPCODE 0x3 part 2

3.6 Binary Operations, OPCODEs 0x4, 0x5, 0x7

These operations are structurally the same, only ALU's input changes in those instructions. Operations' symbols are AND, OR, ADD, respectively. Operations are logical AND (0x4), logical OR (0x5), addition (0x7).

RF_RegSel	*	ARF_RegSel	*	IR_Enable	0	MuxASel	*	ALU_FunSel	*
RF_FunSel	10	ARF_FunSel	10	IR_Funsel	Φ	MuxBSel	*	Mem_WR	0
RF_OutASel	IR(5-4)	ARF_OutCSel	IR(5-4)	IR_LH	Φ	MuxCSel	*	Mem_CS	1
RF_OutBSel	IR(1-0)	ARF_OutDSel	Φ					reset_counter	1

Table 9: Control Unit's control outputs for binary operations in T_2

RF_RegSel and ARF_RegSel are dependent on DESTREG. MuxASel and MuxBSel are

11, connecting output of the ALU. MuxCSel dependent on SRCREG1, if IR(6) = 1 MuxCSel is 1 and connecting RF into ALU, otherwise MuxCSel is 0, connecting ARF into ALU.

ALU_FunSel is dependent on the opcode and for addition it is 0x4, for AND it is 0x7 and for OR it is 0x8.

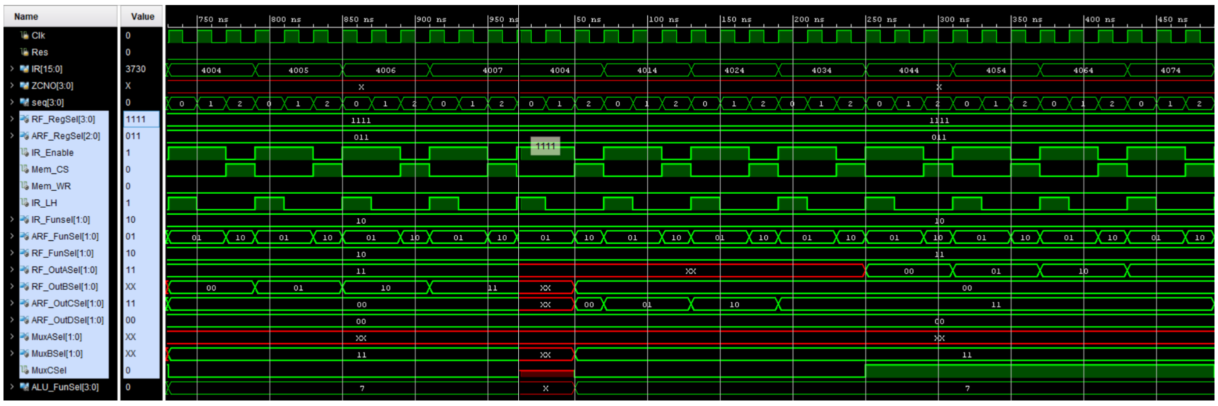


Figure 12: Simulation of OPCODE 0x4 with different SRCREG combinations at T_2

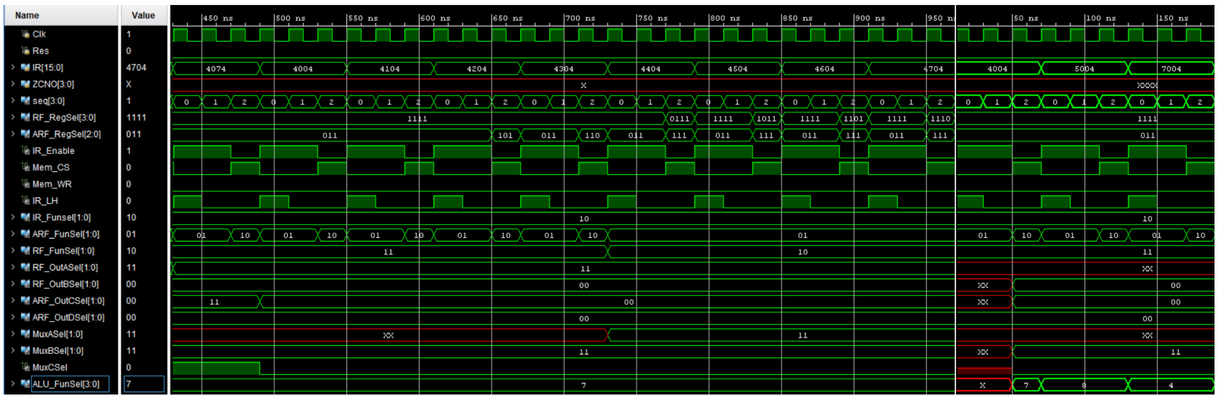


Figure 13: Simulation of binary OPCODEs with different DESTREG combinations at T_2

3.7 Unary Operations, OPCODES 0x6, 0x9, 0xA

These operations are structurally the same, only ALU's input changes in those instructions. Operations' symbols are NOT, LSR, LSL respectively. Operations are logical NOT (0x6), logical right shift (0x9) and logical left shift (0xa).

RF_RegSel and ARF_RegSel are dependent on DESTREG. MuxASel and MuxBSel are 11, connecting output of the ALU. MuxCSel dependent on SRCREG1, if IR(6) = 1 MuxCSel is 1 and connecting RF into ALU, otherwise MuxCSel is 0, connecting ARF into ALU.

RF_RegSel	*	ARF_RegSel	*	IR_Enable	0	MuxASel	*	ALU_FunSel	*
RF_FunSel	10	ARF_FunSel	10	IR_Funsel	Φ	MuxBSel	*	Mem_WR	0
RF_OutASel	IR(5-4)	ARF_OutCSel	IR(5-4)	IR_LH	Φ	MuxCSel	*	Mem_CS	1
RF_OutBSel	Φ	ARF_OutDSel	Φ					reset_counter	1

Table 10: Control Unit's control outputs for unary operations at T_2

ALU_FunSel is dependent on the opcode and for negation it is 0x2, for right shift it is 0xb, for logical left shift it is 0xa.

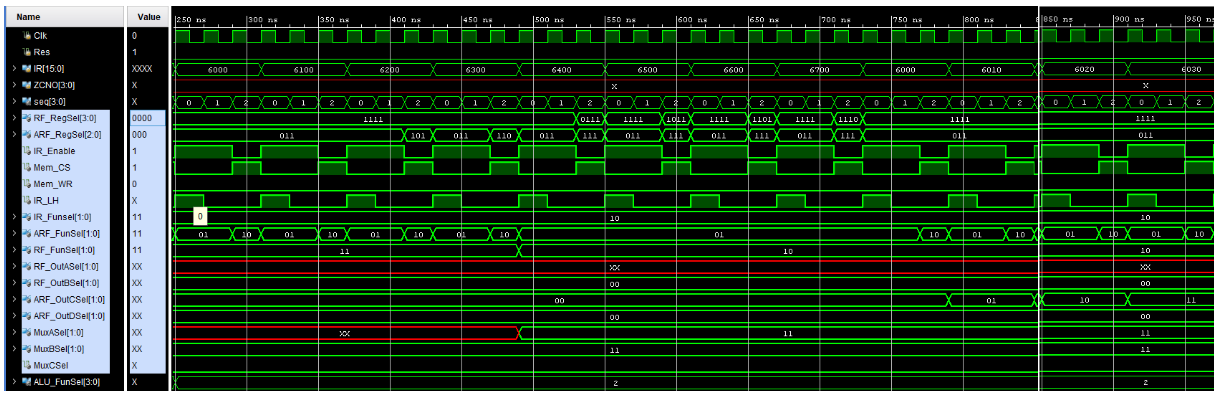


Figure 14: Simulation of OPCODE 0x6 with different register combinations at T_2

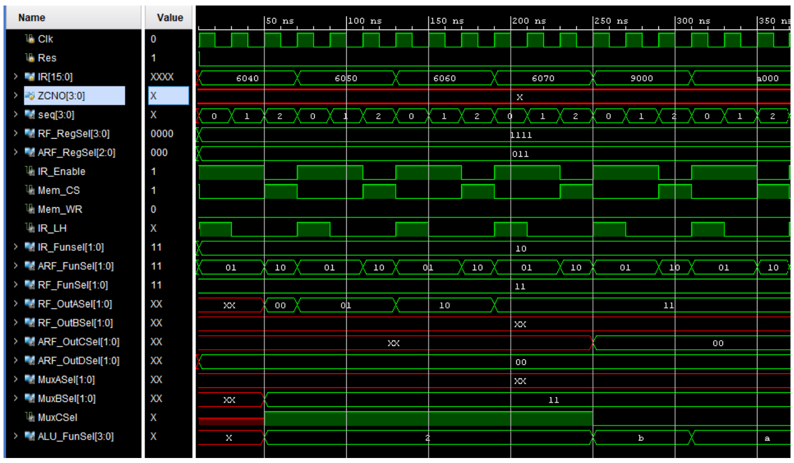


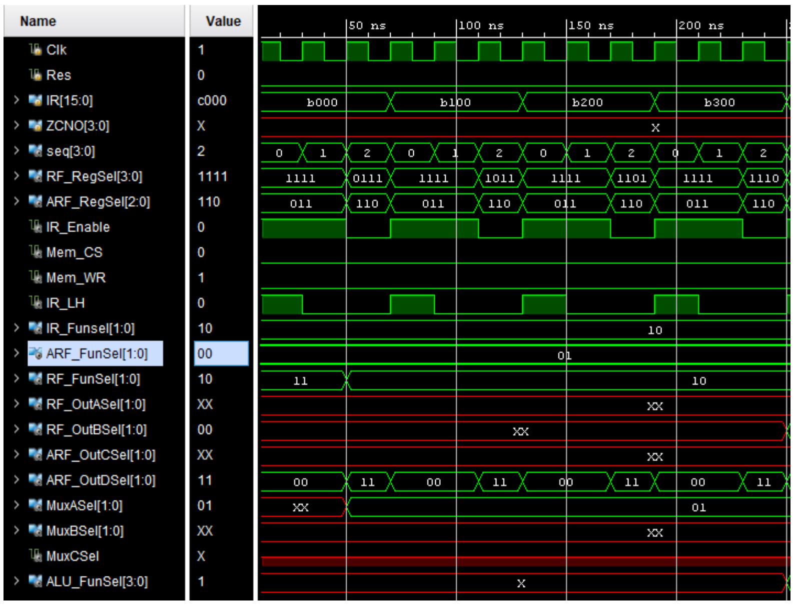
Figure 15: Simulation of unary OPCODEs with different combinations at T_2

3.8 OPCODE 0xB

This operation stands for pulling data from memory where stack pointer points, then decrement the stack pointer. SP is used for stack (LIFO) type structures' calculations. Symbol is PUL and RTL language representation is $SP \leftarrow SP + 1$, $Rx \leftarrow M[SP]$.

RF_RegSel	Rx	ARF_RegSel	110	IR_Enable	0	MuxASel	01	ALU_FunSel	Φ
RF_FunSel	10	ARF_FunSel	01	IR_Funsel	Φ	MuxBSel	Φ	Mem_WR	0
RF_OutASel	Φ	ARF_OutCSel	Φ	IR_LH	Φ	MuxCSel	Φ	Mem_CS	0
RF_OutBSel	Φ	ARF_OutDSel	11					reset_counter	1

Table 11: Control Unit's control outputs for OPCODE 0xB at T_2



Name	Value
Clk	1
Res	0
IR[15:0]	c000
ZCNO[3:0]	X
seq[3:0]	2
RF_RegSel[3:0]	1111
ARF_RegSel[2:0]	110
IR_Enable	0
Mem_CS	0
Mem_WR	1
IR_LH	0
IR_Funsel[1:0]	10
ARF_FunSel[1:0]	00
RF_FunSel[1:0]	10
RF_OutASel[1:0]	XX
RF_OutBSel[1:0]	00
ARF_OutCSel[1:0]	XX
ARF_OutDSel[1:0]	11
MuxASel[1:0]	01
MuxBSel[1:0]	XX
MuxCSel	X
ALU_FunSel[3:0]	1

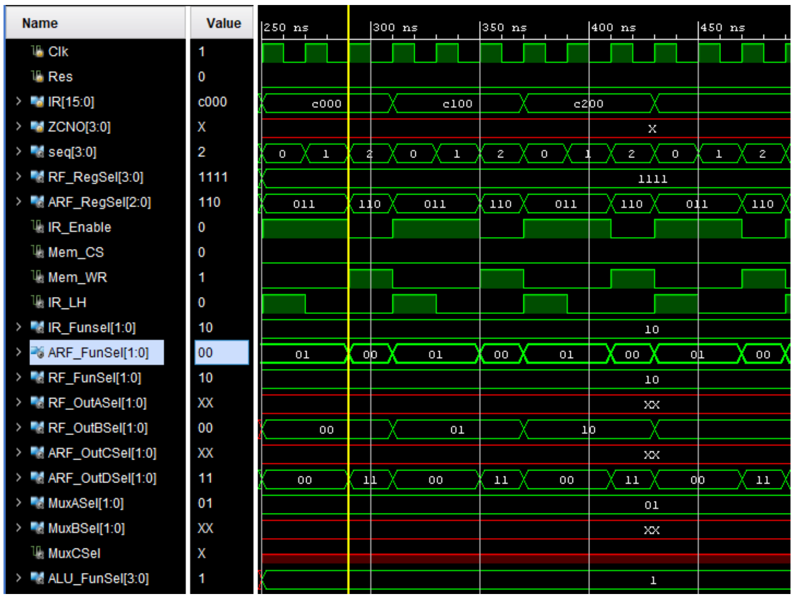
Figure 16: Simulation of OPCODE 0xB with different combinations at T_2

3.9 OPCODE 0xC

This operation stands for pushing data to memory for stack operations. Symbol is PSH and RTL language representation is $M[SP] \leftarrow Rx$, $SP \leftarrow SP - 1$.

RF_RegSel	1111	ARF_RegSel	110	IR_Enable	0	MuxASel	Φ	ALU_FunSel	0x1
RF_FunSel	Φ	ARF_FunSel	00	IR_Funsel	Φ	MuxBSel	Φ	Mem_WR	1
RF_OutASel	Φ	ARF_OutCSel	Φ	IR_LH	Φ	MuxCSel	Φ	Mem_CS	0
RF_OutBSel	Rx	ARF_OutDSel	11					reset_counter	1

Table 12: Control Unit's control outputs for OPCODE 0xC at T_2



Name	Value
Clk	1
Res	0
IR[15:0]	c000
ZCNO[3:0]	X
seq[3:0]	2
RF_RegSel[3:0]	1111
ARF_RegSel[2:0]	110
IR_Enable	0
Mem_CS	0
Mem_WR	1
IR_LH	0
IR_Funsel[1:0]	10
ARF_FunSel[1:0]	00
RF_FunSel[1:0]	10
RF_OutASel[1:0]	XX
RF_OutBSel[1:0]	00
ARF_OutCSel[1:0]	XX
ARF_OutDSel[1:0]	11
MuxASel[1:0]	01
MuxBSel[1:0]	XX
MuxCSel	X
ALU_FunSel[3:0]	1

Figure 17: Simulation of OPCODE 0xC with different combinations at T_2

3.10 OPCODEs 0xD, 0xE

These instructions for moving and incrementing/decrementing the moved data. Since it requires two operations, these instructions are handled in two clock cycles. First clock cycle is identical with MOV except for the reset switch for sequence counter and second instruction is to increment/decrement written data. Symbols for those instructions are INC and DEC. They are very similar, only difference is in their second clock cycle.

RF_RegSel	*	ARF_RegSel	*	IR_Enable	0	MuxASel	Φ	ALU_FunSel	Φ
RF_FunSel	**	ARF_FunSel	**	IR_Funsel	Φ	MuxBSel	Φ	Mem_WR	0
RF_OutASel	Φ	ARF_OutCSel	Φ	IR_LH	Φ	MuxCSel	Φ	Mem_CS	1
RF_OutBSel	Φ	ARF_OutDSel	Φ					reset_counter	1

Table 13: Control Unit's control outputs for OPCODE 0xD or 0xE in T_3

* : Depending on DESTREG. ** : Depending on whether OPCODE is 0xD or 0xE. 01 for increment, 00 for decrement.

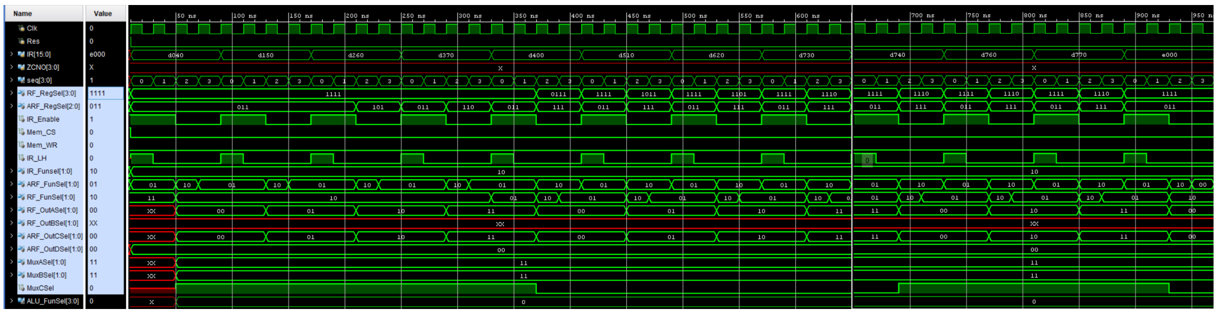


Figure 18: Simulation of OPCODEs 0xD and OxE with different combinations at T_2, T_3

3.11 OPCODE 0xF

This instruction is used for conditional operations, this instruction checks zero flag of the ALU. It's symbol is BNE and its RTL description is $If Z = 0 then PC \leftarrow Value$. In this case value is IR(7-0).

* : ARF_RegSel is 011 if Z = 0, otherwise it is 111.

RF_RegSel	1111	ARF_RegSel	*	IR_Enable	0	MuxASel	Φ	ALU_FunSel	Φ
RF_FunSel	Φ	ARF_FunSel	10	IR_Funsel	Φ	MuxBSel	Φ	Mem_WR	0
RF_OutASel	Φ	ARF_OutCSel	Φ	IR_LH	Φ	MuxCSel	Φ	Mem_CS	1
RF_OutBSel	Φ	ARF_OutDSel	Φ					reset_counter	1

Table 14: Control Unit's control outputs for OPCODE 0xF in T_3

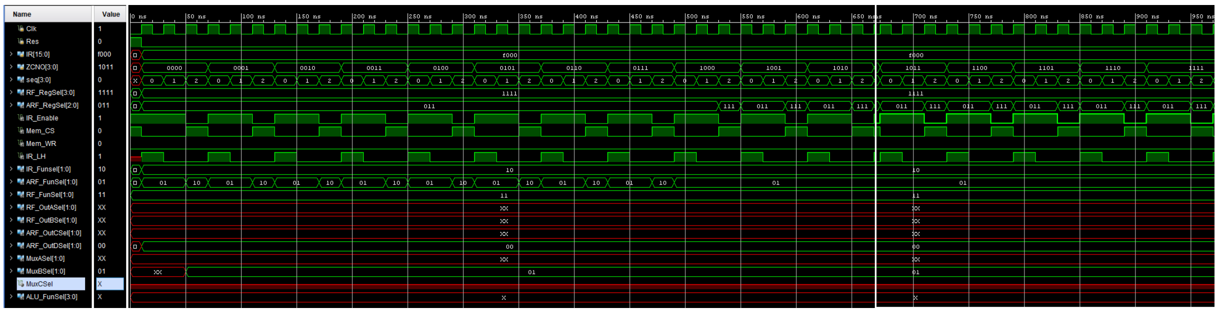


Figure 19: Simulation of OPCODE 0xF with different ZCNO combinations

3.12 OPCODE 0x8

This is the subtraction instruction. This is the longest one (5 clocks long including fetch), because instruction requires $B - A$ ($SRCREG2 - SRCREG1$) but ALU supports $A - B$. Therefore first, Take $A - B$, then take complement, then increment the register. Therefore achieve 2's complement. T_2 is very similar to binary operations, except ALU's fuction signal is 0x6 and reset signal is 0. Then in T_3, control signals are very similar to unary operation NOT, except SRCREG is DESTREG and reset signal is 0. And lastly in T_4 signal is similar to increment operation. $DESTREG \leftarrow DESTREG + 1$.

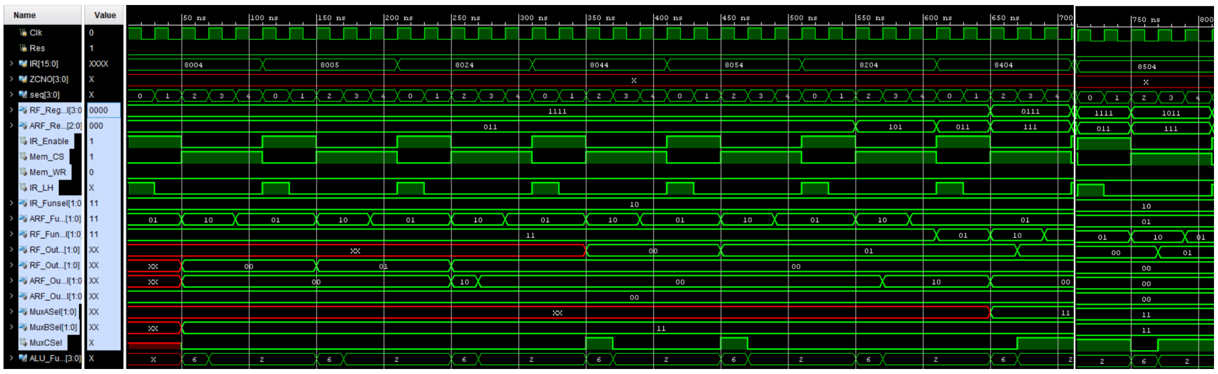


Figure 20: Simulation of OPCODE 0x8 with different source and destinations.

4 Overall Design and Connecting Control System

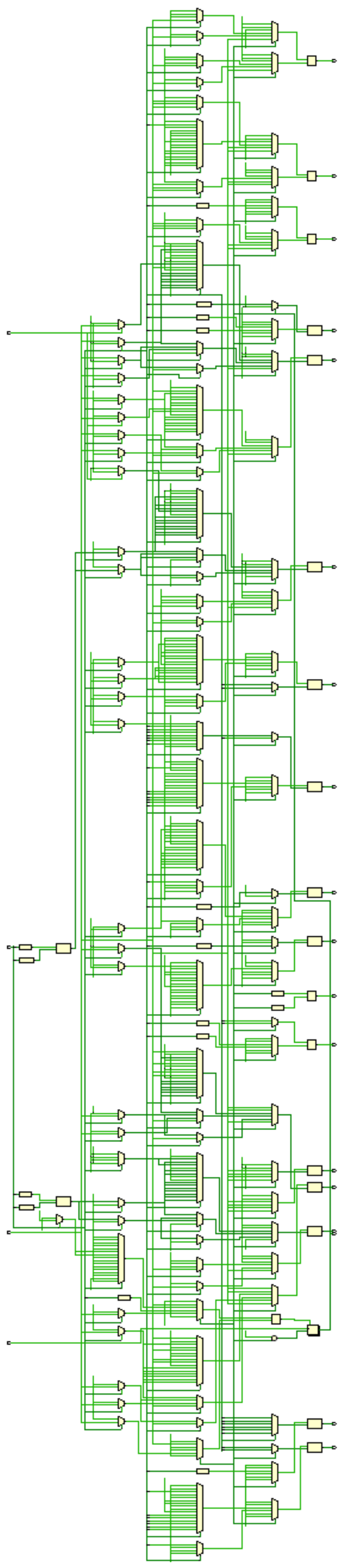


Figure 21: Overall design of the control unit.

4.1 Connecting Systems Together

Control unit is for ALU System that we have designed in the first project. To be able to make it work correctly, we must connect those systems together. Also we can write small programs in memory.

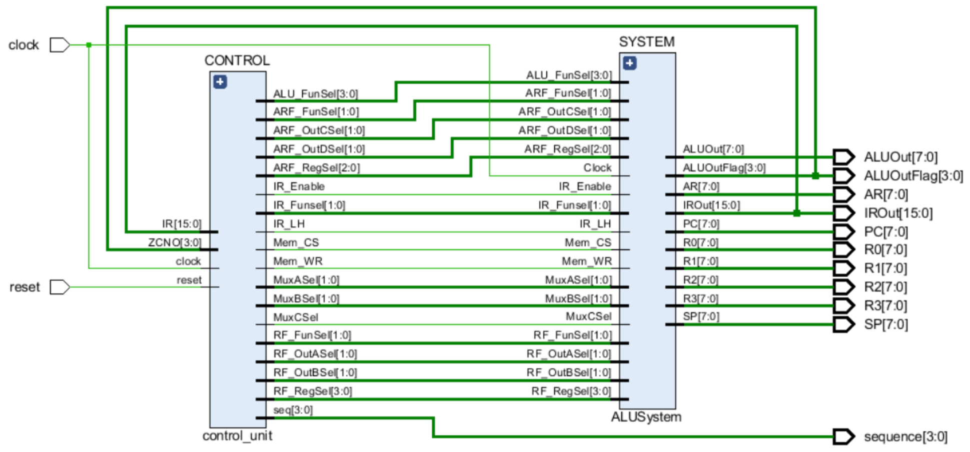
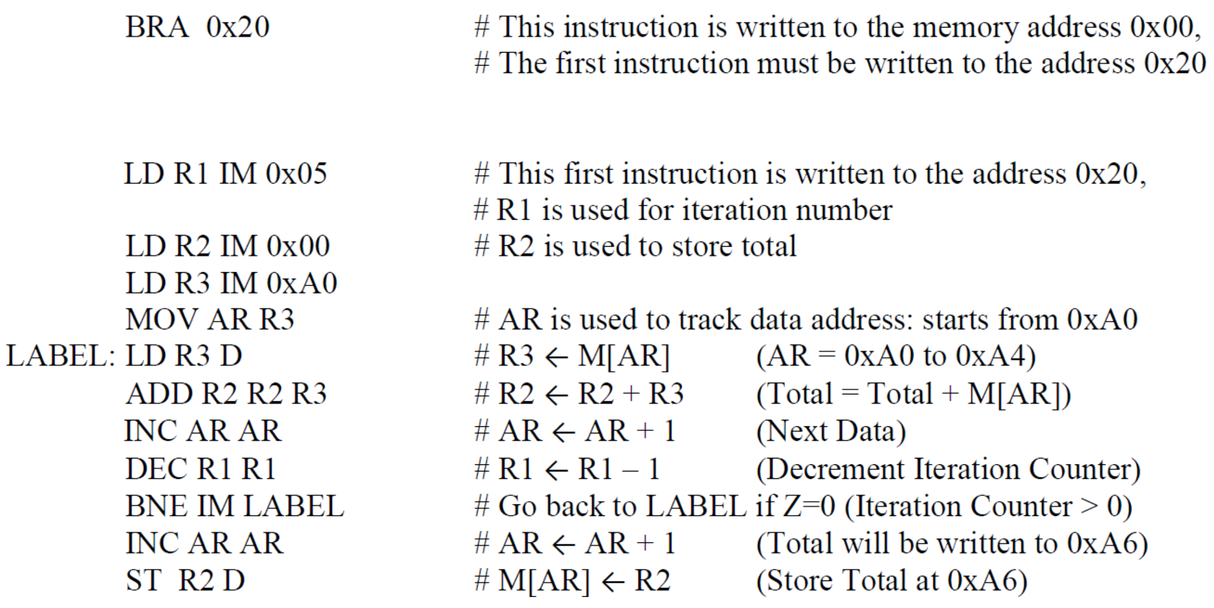


Figure 22: Connected system with ALU System and Control Unit.

4.2 Running a Program

In the homework paper, there was an example program, we have written the program into memory. Memory file is also included in our homework submission.



```
          BRA  0x20          # This instruction is written to the memory address 0x00,
                             # The first instruction must be written to the address 0x20


          LD R1 IM 0x05      # This first instruction is written to the address 0x20,
                             # R1 is used for iteration number
          LD R2 IM 0x00      # R2 is used to store total
          LD R3 IM 0xA0
          MOV AR R3          # AR is used to track data address: starts from 0xA0
LABEL:    LD R3 D            # R3 ← M[AR]          (AR = 0xA0 to 0xA4)
          ADD R2 R2 R3       # R2 ← R2 + R3        (Total = Total + M[AR])
          INC AR AR          # AR ← AR + 1         (Next Data)
          DEC R1 R1          # R1 ← R1 – 1         (Decrement Iteration Counter)
          BNE IM LABEL       # Go back to LABEL if Z=0 (Iteration Counter > 0)
          INC AR AR          # AR ← AR + 1         (Total will be written to 0xA6)
          ST  R2 D           # M[AR] ← R2          (Store Total at 0xA6)
```

Figure 23: Proposed program [1]

Unfortunately I encountered some problems while program was running, I fixed those issues and then program worked fine. The operands inside the memory addresses are:

Memory Address	0xA0	0xA1	0xA2	0xA3	0xA4
Value Stored	1	2	3	4	5

Table 15: Values stored in the memory slots

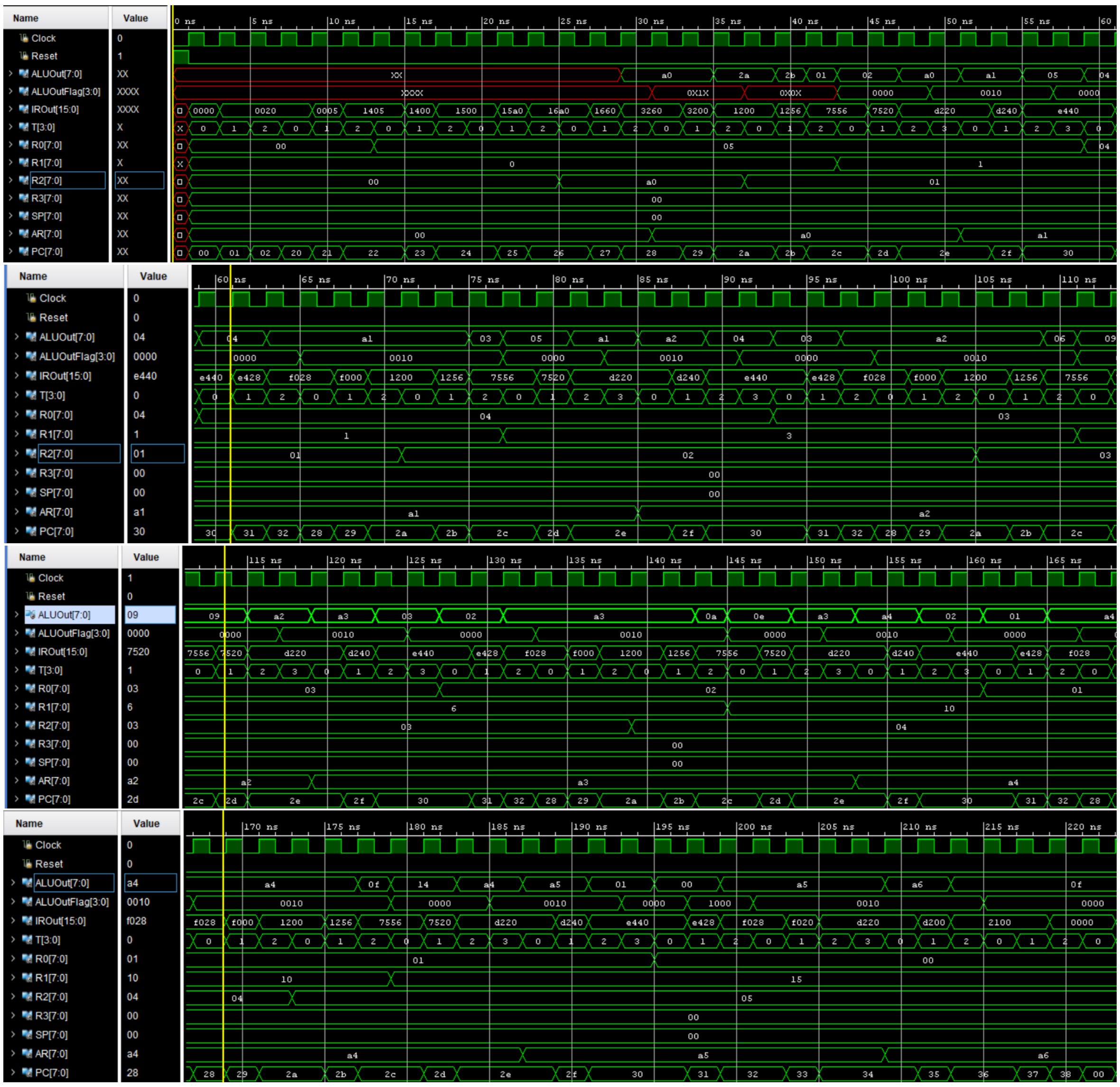


Figure 24: Simulation of the program.

5 DISCUSSION AND CONCLUSION

Since the circuit was complex and had many multiplexers, it was difficult and complicated to create a control circuitry. This complexity made it able to do many of the operations quite fast (mostly in 3 or 4 clock cycles including fetching, only in one instance 5 clock cycles) but together brought its complexity. A simpler design would be much easier to handle but it would probably work slower.

One problem with the design was OPCODE had unused slots and also due to memory connection's limitations fetching was handled in 2 clock cycles. This problem would be solved if memory word size was 16-bits long. Also our system had much more capabilities beyond the limitations from OPCODE, such as increment/decrement operations on registers, setting them to zero etc. In addition we didn't use ALU flags' full potential.

In conclusion we have used our Verilog skills and Computer Organization course knowledge to create a basic working computer.

REFERENCES

[1] Istanbul Technical University Computer Engineering Department. Computer organization. *BLG 222E Computer Organization Assignment Sheets*, 2(3):1–7, May 2022.