ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION
PROJECT 2 REPORT

CRN : 21335

LECTURER : Prof. Dr. Deniz Turgay Altılar

GROUP MEMBERS:

150230734 : TAŞKIN ÖKMEN

SPRING 2024

Contents

1 INTRODUCTION

In the second project of BLG222E Computer Organization we have used Verilog Hardware Description language to implement a ,part of CPU, the Control Unit. The design has consist of a hardwired control unit with address reference and register based instructions. By combining all the essential parts of ALU and control unit we can simulate the basic computer.

2 MATERIALS AND METHODS

2.1 TASK DISTRIBUTION

The whole project including the implementation of the system modules (CPUSystem, counter, control unit, Register, IR, RF, ARF, ALU, ALUSystem) in verilog, debugging and the report is done by TAŞKIN ÖKMEN 150230734. Therefore, TAŞKIN ÖKMEN is the group representative.

2.2 PARTS OF THE PROJECT

2.3 SEQUENCE COUNTER

The sequence counter is used to synchronize and divide complex instructions into simple ordered microoperations. It trigger with positive edge clock signal if enabled and counter one by one between 0 - 16 in binary. Furthermore, in order to generate timing signals individually, a 4:64 decoder is used. The schematic of the sequence counter design is given below.

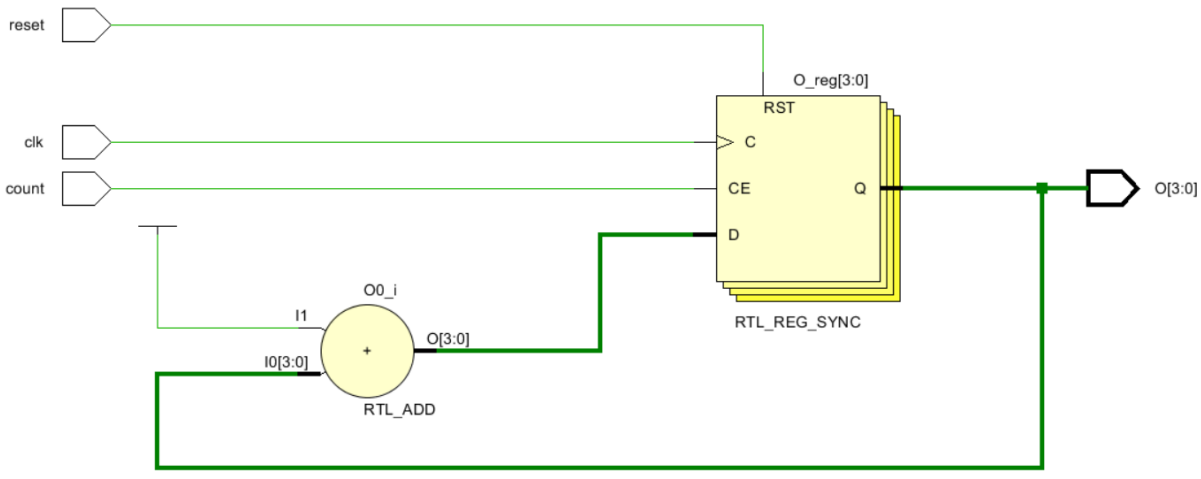

Figure 1: Sequence Counter schematic

2.4 FETCH AND DECODE PHASES

2.4.1 FETH LOW AND HIGH PHASE

The instructions are stored in memory in little-endian order. The RAM has an 8-bit output, therefore the instruction register cannot be filled in one clock cycle. Hence, we have to load MSB and LSB in 2 clock cycles. In T0 the LSB of the instruction is loaded from an address and for T1 the MSB of the instruction is loaded from an address to instruction register. Furthermore, we have to increment the program counter at T0 and T1 therefore, it can continue to fetch the next instructions. The schematic of the fetch phase design is given below.

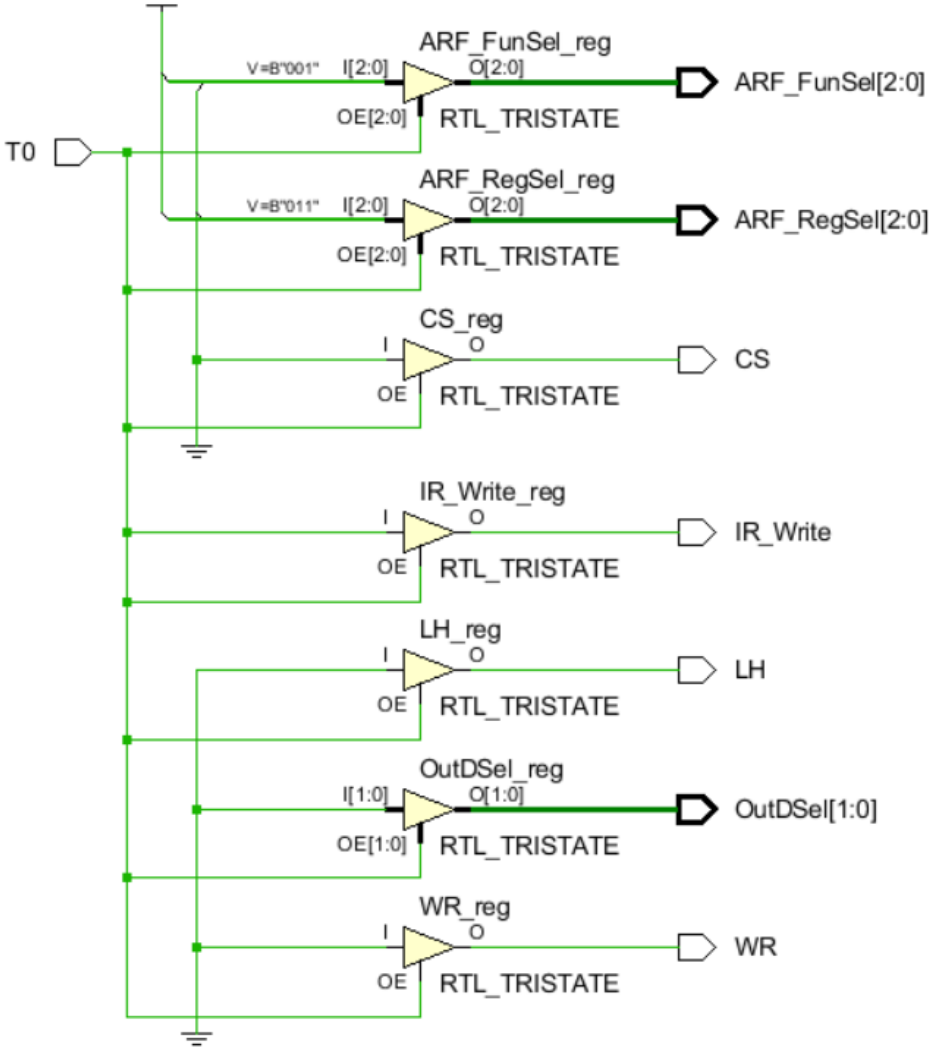

Figure 2: Fetch high phase design

The fetch high phase is given below. It reads from memory and alters the MSB (15-8) bits of instruction register.

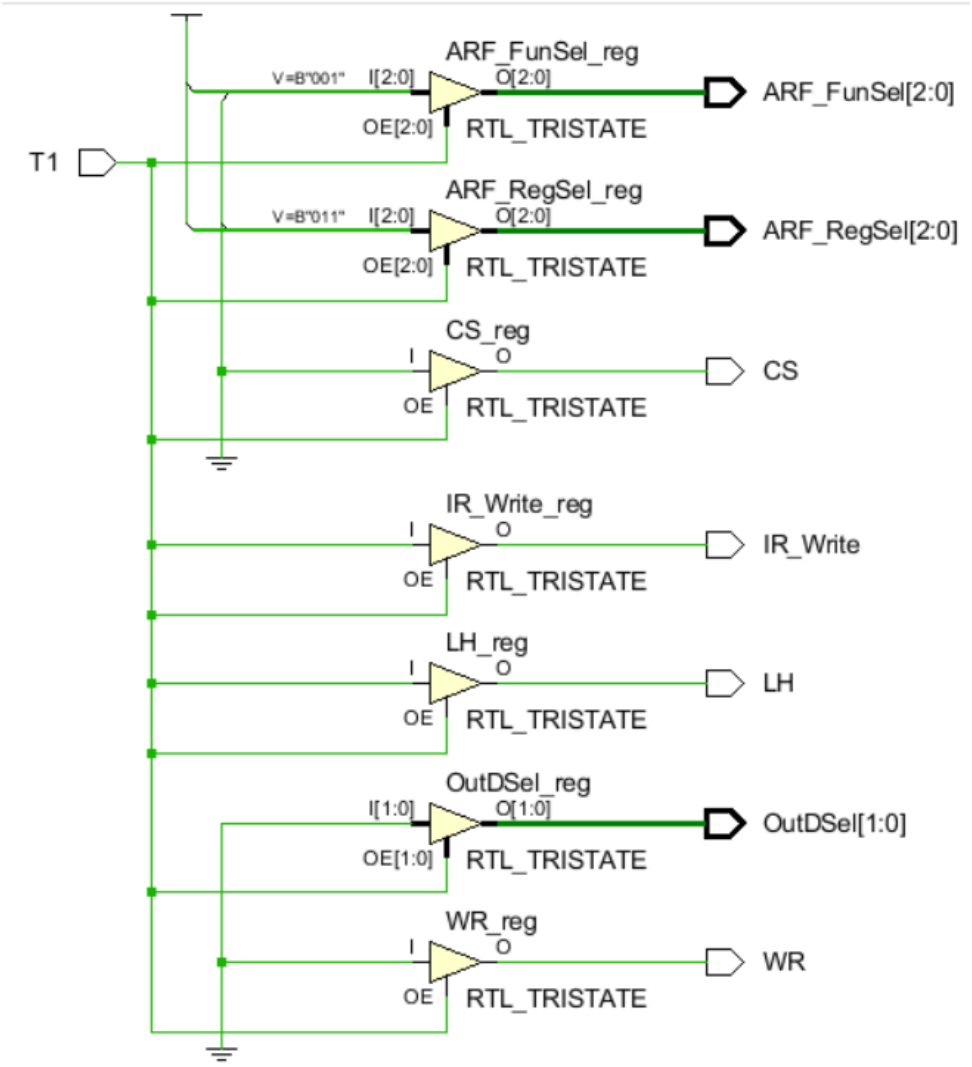

Figure 3: Fetch low phase design

2.4.2 DECODE PHASE

There were two types of instructions, address reference and register based instructions. Furthermore, for 34 instructions in order to generate discrete signals we have used 6:64 decoder. The input of decoder is the MSB 6 bits (15 - 10) of instruction the operation code. Therefore, we have covered the 34 instructions with 64 separate signals. Also, in the decode phase other relevant parts of the instruction: source-destination registers and address are read and stored.

2.5 CONTROL UNIT

The control unit module takes the inputs of arithmetic logic unit system (FunSel, RegSel, MUXSel...) and generates relevant control signals. The schematic of the control unit design is given below.

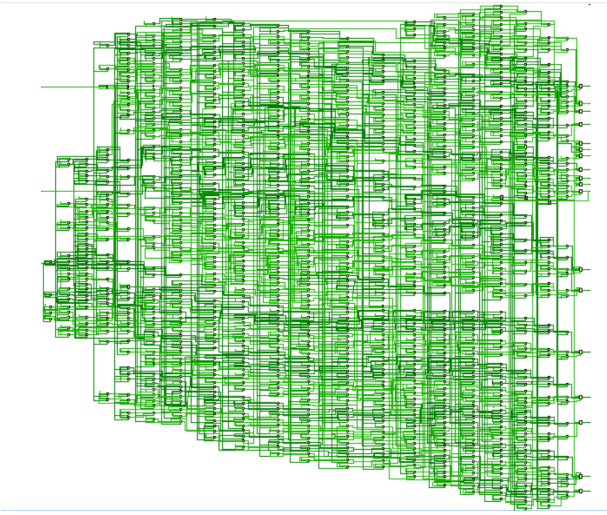

Figure 4: Control Unit design

2.6 CPU SYSTEM

At last, the CPUSystem module, consist of both the control unit and arithmetic logic unit system modules. It connects and completes the final version of the CPU. The schematic of the CPUSystem design is given below.

2.7 IMPLEMENTATION OF INSTRUCTIONS

There were two types of instructions, address reference and register based instructions. In order to generate relevant control signals for instructions we have used register transfer language to reduce the complex operations into simple ordered ones. Furthermore, the register based instructions source and destination register can be located in either register file or address register file. Therefore, relevant control signals must be generated for to select corresponding FunSel and RegSel inputs of RF and ARF. The following RTL sequence shows the fetch phase.

$$T0 : IR[7:0] \leftarrow Mem[PC], PC \leftarrow PC + 1$$

$$T1 : IR[15:8] \leftarrow Mem[PC], PC \leftarrow PC + 1$$

The control signals of fetch phase should be ARFOutDSel: 2'b00, MemCS: 1'b0, MemWR: 1'b0, IRLH: 1'b0, IRWrite: 0'b1 for reading from memory and writing to instruction register and ARFRegSel: 3'b011, ARFFunSel: 3'b001 for incrementing the program counter.

RTL of 0x00-BRA: PC ¡- PC + VALUE instruction is given below. Sequence counter is reseted end of the instruction.

$$D_0T2 : S1 \leftarrow IR[7:0]$$

$$D_0T3 : S2 \leftarrow PC$$

$$D_0T4 : PC \leftarrow S1 + S2$$

$$D_0T5 : SC \leftarrow 0$$

RTL of 0x03-POP: SP ← SP + 1, Rx ← M[SP] instruction is given below.

$$D_3T2 : R_x \leftarrow Mem[SP]$$

$$D_3T3 : SP \leftarrow SP + 1$$

$$D_3T4 : SC \leftarrow 0$$

RTL of 0x05-INC: DESTRG ¡- SREG1 + 1 instruction is given below.

$$D_5T2 : DSTREG \leftarrow SREG1$$

$$D_5T3 : DSTREG \leftarrow DSTREG + 1$$

$$D_5T4 : SC \leftarrow 0$$

RTL of 0x06-DEC: DESTRG ¡- SREG1 + 1 instruction is given below.

$$D_6T2 : DSTREG \leftarrow SREG1$$

$$D_6T3 : DSTREG \leftarrow DSTREG - 1$$

$$D_6T4 : SC \leftarrow 0$$

Another important part for register based instructions was selecting correct registers from register file or address register file. For example, RTL of 0x0F-XOR instruction is given below. At T2, if SREG1 is in ARF (SREG1[2] = 0) SREG1 is copied to scratch register S1, after that at T3 for SREG1 it is also applied. The control signals are MuxASel = 2'b01, RFScrSel = 4'b0111, RFFunSel = 3'b010 for copying SREG1,2 to S1. At T4, in order to determine the location of DSTREG, first logic operation is performed ALUFunSel = 5'b11001, then if DSTREG is in RF (DSTREG[2] = 1), ALUOut copied to RF register MuxASel = 2'b00, RFFunSel = 3'b010, then DSTREG is in ARF, ALUOut copied to ARF register MuxBSel = 2'b00, ARFFunSel = 3'b010.

if SREG1[2] = 0 or SREG2[2] = 0 (SREG1,2 is in ARF)

$$D15T2SREG1[2]' : S1 \leftarrow SREG1$$

$$D15T3SREG2[2]' : S2 \leftarrow SREG2$$

$$D15T4 : ALUOut \leftarrow S1XORS2$$

if DSTREG[2] = 1 (DSTREG is in RF)

$$D15T4DSTREG[2] : DSTREG \leftarrow ALUOut$$

then (DSTREG is in ARF)

$$DSTREG \leftarrow ALUOut$$

RTL of 0x16-ADD: DESTRG ¡- SREG1 + SREG2 instruction is given below.

if SREG1[2] = 0 or SREG2[2] = 0 (SREG1,2 is in ARF)

$$D22T2SREG1[2]' : S1 \leftarrow SREG1$$

$$D22T3SREG2[2]' : S2 \leftarrow SREG2$$

$$D22T4 : ALUOut \leftarrow S1 + S2$$

if DSTREG[2] = 1 (DSTREG is in RF)

$$D22T4DSTREG[2] : DSTREG \leftarrow ALUOut$$

then (DSTREG is in ARF)

$$DSTREG \leftarrow ALUOut$$

3 RESULTS OF THE SIMULATIONS

We have tested our system to see if it performs the operations properly. The simulation files have helped us to debug and take actions for the deficient parts of our system. The following figures are the results of the given simulation files in Vivado respectively.

3.1 Fetch Phase

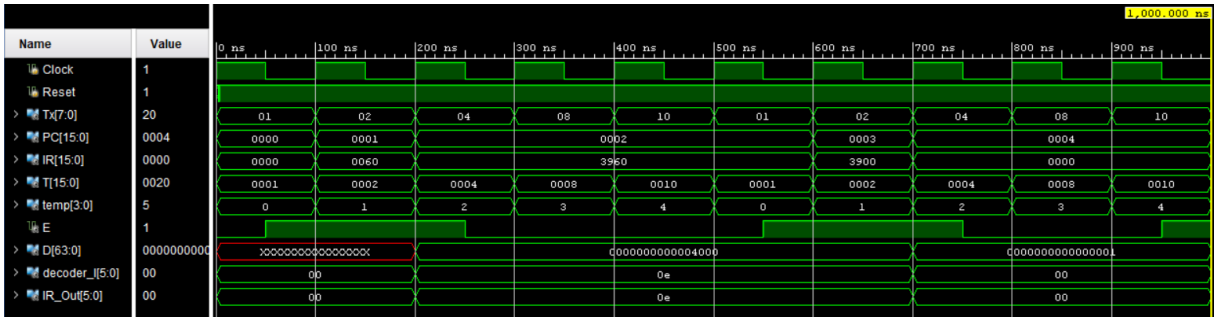

Figure 5: Simulation of fetch phase

3.2 LDR Instruction

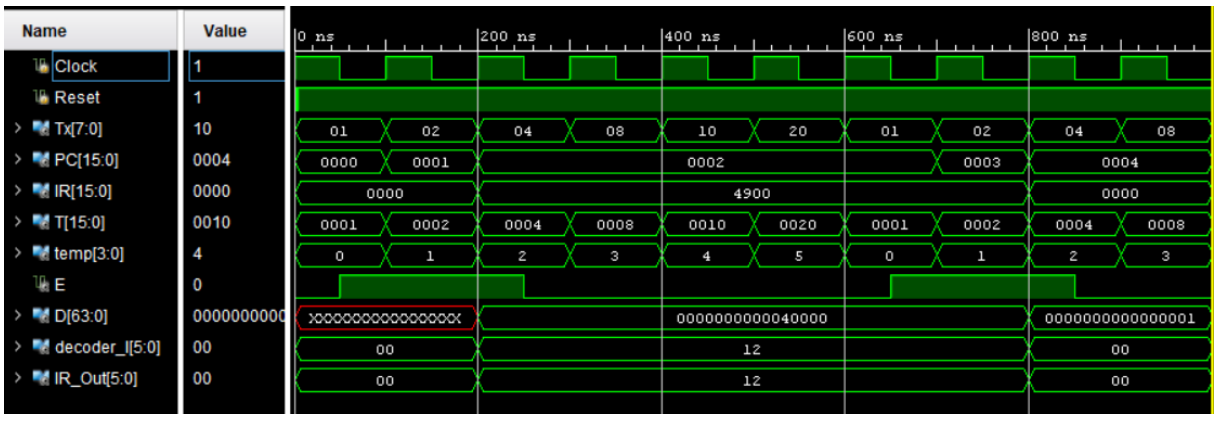

Figure 6: Simulation of LDR

IR was set to 0x4900 meaning load to R2 (DSTREG = M[AR]). AR was previously set to 0x00FD, and in memory M[0x00FD] = 0xAA, M[0x00FE] = 0xBB. In the end of the simulation, as LDR instruction takes in place the value of R2 changed to 0xBBAA.

```
Output Values:
T:  32
Address Register File: PC:     2, AR:   256, SP:   253
Instruction Register : 18688
Register File Registers: R1: 43690, R2: 48042, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x
```

Figure 7: Output of LDR

3.3 MOVS Instruction

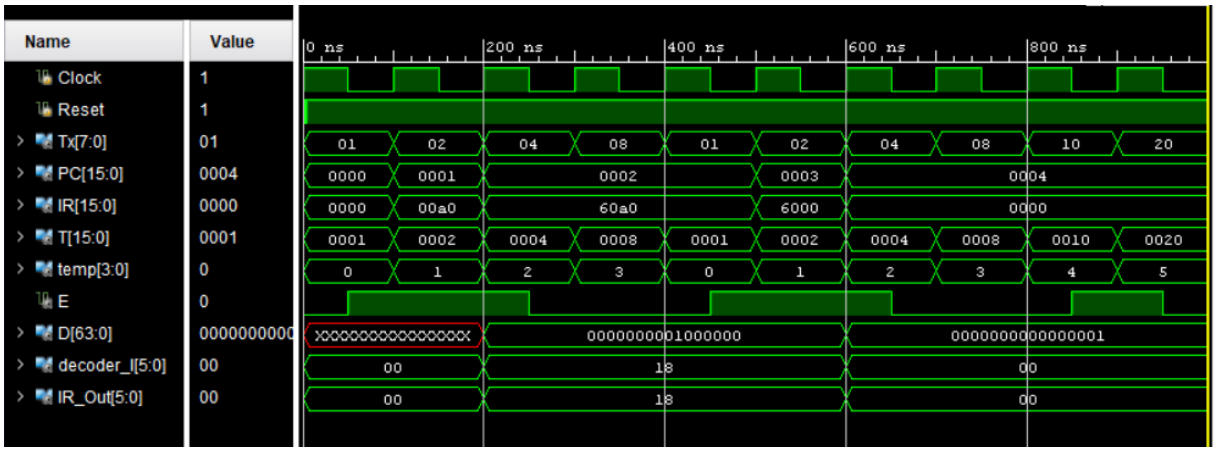

Figure 8: Simulation of MOVS

IR was set to 0x60A0 meaning move R1 to SP. SP was previously set to 0x0 and R1 was 0xAAAA. In the end of the simulation, as MOVS instruction takes in place the value of SP changed to 0xAAAA.

```
Output Values:
T:   8
Address Register File: PC:     2, AR:   253, SP: 43690
Instruction Register : 24736
Register File Registers: R1: 43690, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     0, S2:     0, S3:     0, S4:     0
ALU Flags: Z: 0, N: 0, C: 1, O: 0
ALU Result: ALUOut: 43690
```

Figure 9: Output of MOVS

4 DISCUSSION

The project overall required dedication to implement the all the parts. It required the understanding of correlation and relevance between successive parts. First we have implemented the sequence counter for generating timing signals. After that, we have implemented decoder module 6:64 decoder for operation code and 4:16 decoder for timing signals. Then, for the control unit we have programmed 34 instructions in order. At last, we have connected the control unit module with ALUSystem to form the CPUSystem module. Overall, we can confidently say we have an basic understanding of the CPU system and control unit.

5 CONCLUSION

The most difficult part was determining the control signals for 34 instructions. It required attention. The simulations files provided an insight, however it was exhaustive and repetitive to observe and interpret the input and output signals at Xilinx Vivado's simulation. It was difficult to grasp and understand the whole system at the tests. In breif, we can say, we have learned the general system of basic computer.

REFERENCES

John Doe. Computer organization. An example journal, 22(4):10–16, February 2020.