ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION
PROJECT 1 REPORT

CRNs : 21334

21336

LECTURERs : GÖKHAN İNCE

MUSTAFA ERSEL KAMAŞAK

GROUP MEMBERS:

150210087 : HİLAL KARTAL

150210100 : SELİN YILMAZ

SPRING 2024

Contents

1 INTRODUCTION [10 points]

In this project, Verilog hardware description language is used to implement a basic computer. First, we implemented modules for 16-bit *registers* (small storage units providing fast access to frequently used data) which has different functionalities. In second part, we designed an *Instruction Register* with input 8-bit and output 16-bit. Instruction register holds instructions in itself. We designed a *Register File* (an array consists of registers in a central processing unit(CPU)) which includes multiple registers. And we designed an *Address Register File* (it has program counter (PC), address register (AR), and stack pointer (SP)) which includes three 16-bit address registers. In the third part, we designed an *Arithmetic Logic Unit (ALU)* (a combinational digital circuit for performing bitwise and arithmetic operations). In the last part, we combined them using multiplexers and acquired *Arithmetic Logic Unit System*

1.1 Task Distribution

We distributed the parts in the following way.

- Part 1 done by Selin
- Part 2a done by Hilal
- Part 2b done by Hilal and Selin
- Part 2c done by Hilal and Selin
- Part 3 done by Hilal and Selin
- Part 4 done by Hilal and Selin
- Report/Introduction done by Selin
- Report/Methods done by Hilal
- Report/Results done by Hilal
- Report/Discussion done by Selin
- Report/Conclusion done by Selin

2 MATERIALS AND METHODS [40 points]

2.1 Materials

1. Verilog

2.2 Methods

2.2.1 PART - 1

In the this part we were asked to design a 16-bit general purpose register that has 8 functionalities which are controlled by a 3-bit function selector signal (FunSel) and an enable input(E). Function table can be seen from Table 1

E	FunSel	Q^+
0	ϕ	Q(Retain value)
1	000	Q-1 (Decrement)
1	001	Q+1 (Increment)
1	010	I (Load)
1	011	0 (Clear)
1	100	Q's (15-8) bits are 0, Q's (7-0) bits are I's (7-0) bits (Write Low)
1	101	Q's (15-8) bits are Q's(15-8),Q's (7-0) bits are I's (7-0) bits. (Only Write Low)
1	110	Q's (15-8) bits are I's (7-0) bits, Q's (7-0) bits are Q's(7-0)(Only Write High)
1	111	Q (15-8) ← Sign Extend (I (7)), Q (7-0) ← I (7-0) (Write Low)

Table 1: FunSel Table for Register

Our design can be seen from Figure 1.

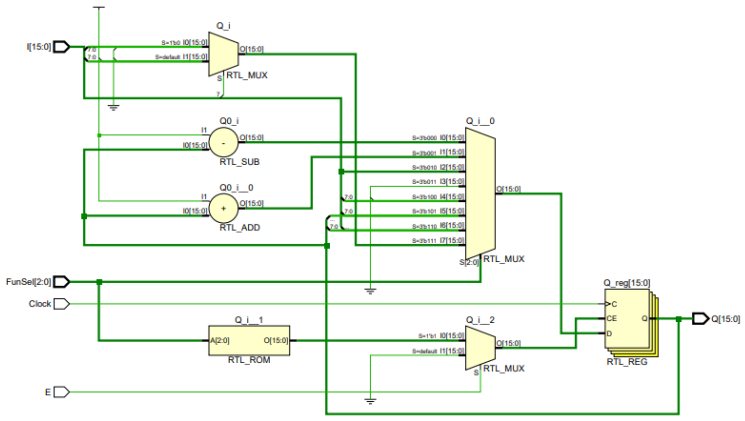

Figure 1: Schematic of 16-bit general purpose register

2.2.2 PART - 2

We are asked to implement a register file that includes an instruction register (IR), general purpose and scratch registers, and address register file (ARF) that includes program counter (PC), address register (AR), and stack pointer (SP).

1. PART-2A: Instruction Register File
   Functionalities of this register can be seen from Table 2.

L'H	Write	IR^+
ϕ	0	IR does not change.(Retain value)
0	000	IR's (15-8) bits are 0, IR's (7-0) bits are I. (Load LSB)
1	001	IR's (15-8) bits are I, IR's (7-0) bits are 0. (Load MSB)

Table 2: L'H and Write Table for Instruction Register

Our design can be seen from Figure 2.

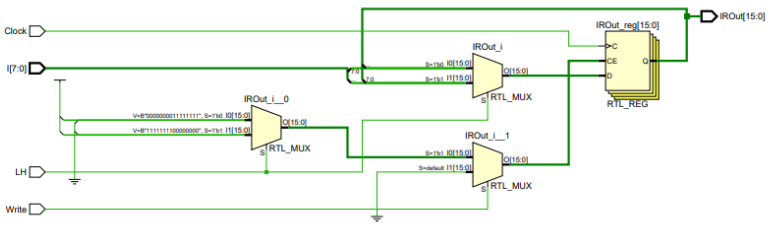

Figure 2: Schematic of 16-bit Instruction Register

2. PART-2B: Register File

   (a) This system includes 8 of the general purpose registers we designed at the Part 1.

   (b) Four of them are used as general purpose registers while the other four is used as scratch registers.

   (c) Regiser selection is made with RegSel and ScrSel inputs.

      i. RegSel[3] = 0 means R1 is Enabled.
      RegSel[2] = 0 means R2 is Enabled.
      RegSel[1] = 0 means R3 is Enabled.
      RegSel[0] = 0 means R4 is Enabled.

      ii. ScrSel[3] = 0 means S1 is Enabled.
      ScrSel[2] = 0 means S2 is Enabled.
      ScrSel[1] = 0 means S3 is Enabled.
      ScrSel[0] = 0 means S4 is Enabled.

   (d) As we are using the same registers we designed at part 1, it can be seen that our Funsel works the same for the selected register. (Table 1)

Output select of this system can be seen from Tables 3 and 4.

OutASel	OutA
000	R1's output is sent to OutA.
001	R2's output is sent to OutA.
010	R3's output is sent to OutA.
011	R4's output is sent to OutA.
100	S1's output is sent to OutA.
101	S2's output is sent to OutA.
110	S3's output is sent to OutA.
111	S4's output is sent to OutA.

Table 3: OutASel Table

OutBSel	OutB
000	R1's output is sent to OutB.
001	R2's output is sent to OutB.
010	R3's output is sent to OutB.
011	R4's output is sent to OutB.
100	S1's output is sent to OutB.
101	S2's output is sent to OutB.
110	S3's output is sent to OutB.
111	S4's output is sent to OutB.

Table 4: OutBSel Table

Our design can be seen from Figure 3.

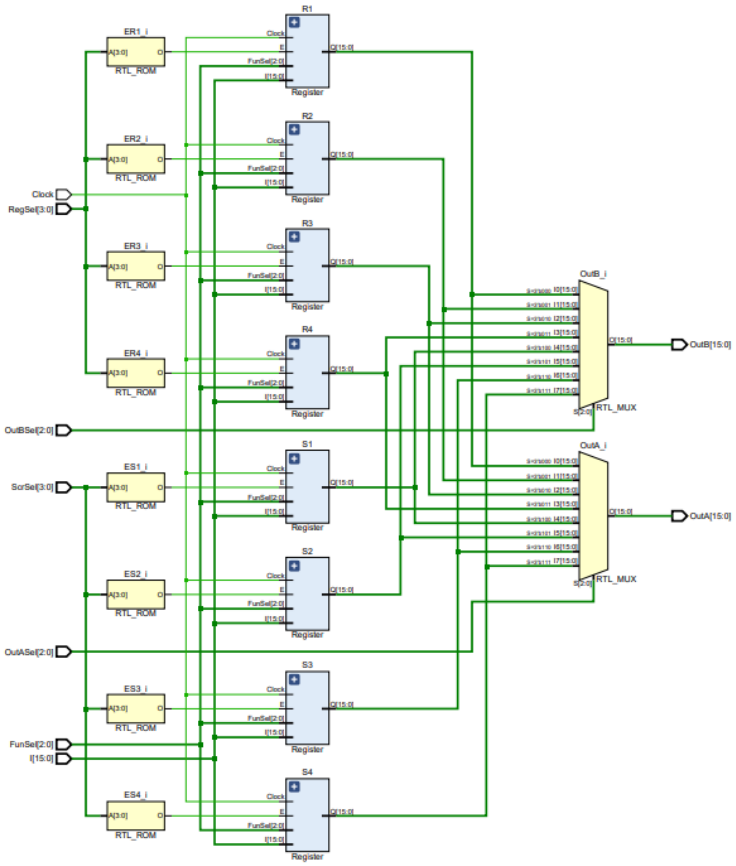

Figure 3: Schematic of Register File that includes general purpose registers and scratch registers

3. PART-2C: Address Register File

   (a) This system includes 3 of the general purpose registers we designed at the Part 1.

   (b) They are used as program counter (PC), address register (AR), and stack pointer (SP).

   (c) Regiser selection is made with 3 bit RegSel input.
   RegSel[2] = 0 means PC is Enabled.
   RegSel[1] = 0 means AR is Enabled.
   RegSel[0] = 0 means SP is Enabled.

   (d) As we are using the same registers we designed at part 1, it can be seen that our Funsel works the same for the selected register. (Table 1)

OutCSel	OutC
00	PC's output is sent to OutC.
01	PC's output is sent to OutC.
10	AR's output is sent to OutC.
11	SP's output is sent to OutC.

Table 5: OutCSel Table

OutDSel	OutD
00	PC's output is sent to OutD.
01	PC's output is sent to OutD.
10	AR's output is sent to OutD.
11	SP's output is sent to OutD.

Table 6: OutDSel Table

Our design can be seen from Figure 4.

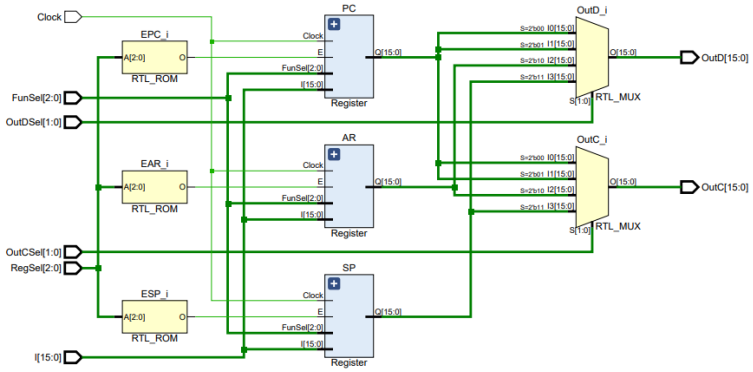

Figure 4: Schematic of address register file

2.2.3 PART - 3

We are asked to implement an Arithmetic Logic Unit (ALU) that takes two 16 bits inputs and does the operation that is selected via 5 bit FunSel input. This ALU also includes a 4 bit FlagsOut register that keeps information on:

1. Whether the Output is 0 (Z). If it is FlagsOut[3] is set to 1.
2. Whether operation yield a Carry Out (C). If did FlagsOut[2] is set to 1.
3. Whether Output is Negative (N). If it is Flagsout[1] is set 1.
4. Whether operation yield a overflow (O). If it did FlagsOut[0] is set to 1.

FlagsOut register also has a Enable input WF. If WF is 0 the register does not change it's state even if one of the upper conditions hold.

FunSel[4] gives the information on whether the operation is done on 8 or 16 bits.

1. FunSel[4] = 0 means it is an 8 bit operation and ALUOut's (15-8) bits will be loaded 0.
2. FunSel[4] = 1 means it is an 16 bit operation.

Function Selection and it's affect can be seen from Table 7. (It should be noted that this table only covers (3-0) bits of FunSel as MSB only gives information on operation size.)

FunSel[3:0]	ALUOut	Z	C	N	O
0000	A	Affected	NOT	Affected	NOT
0001	B	Affected	NOT	Affected	NOT
0010	$\overline{A}$	Affected	NOT	Affected	NOT
0011	$\overline{B}$	Affected	NOT	Affected	NOT
0100	A + B	Affected	Affected	Affected	NOT
0101	A + B + FlagsOut[2]	Affected	Affected	Affected	NOT
0110	A - B (A + $\overline{B}$ + 1)	Affected	Affected	Affected	NOT
0111	A & B	Affected	NOT	Affected	NOT
1000	A| B	Affected	NOT	Affected	NOT
1001	A^B	Affected	NOT	Affected	NOT
1010	$\overline{A\&B}$	Affected	NOT	Affected	NOT
1011	Logical Shift Left A	Affected	Affected	Affected	NOT
1100	Logical Shift Right A	Affected	Affected	Affected	NOT
1101	Arithmetic Shift Right A	Affected	Affected	NOT	NOT
1110	Circular Shift Left A	Affected	Affected	Affected	NOT
1111	Circular Shift Right A	Affected	Affected	Affected	NOT

Table 7: FunSel Table for ALU

Our design can be seen from Figure 5.

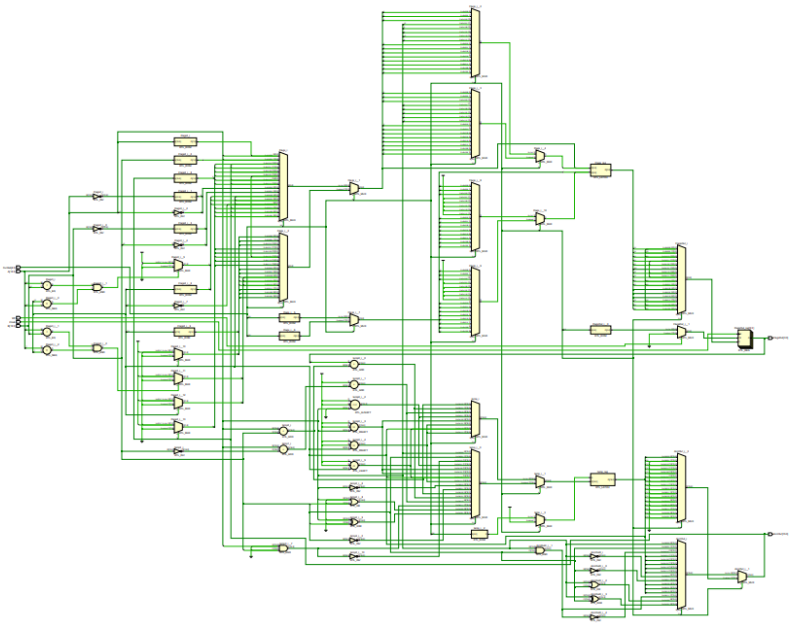

Figure 5: Schematic of Arithmetic Logic Unit

2.2.4 PART - 4

In this section we are asked to combine all the previous parts and Memory (pre-designed and given with the assignment) to make an Arithmetic Logic Unit System. To combine all the parts we use 3 Multiplexers.
Combination goes as:

1. Register File's outputs connects to ALU.
2. ALU's output connects to MUX's (A,B,C).
3. MUX A's output connects to Register File.
4. MUX B's output connects to Address Register File.
5. MUX C's output connects to Memory.
6. Address Register File's output connects to Memory, MUX A and MUX B .
7. Memory's output connects to MUX A, MUX B and Instruction Register.
8. Instruction Register's output connects to MUX A and MUX B.

MUX Output Selections can be seen from Tables 8, 9 and 10.

MuxASel	MuxAOut
00	ALU's Output
01	Address Register File's Output
10	Memory's Output
11	Instruction Register's (7-0) bits

Table 8: MuxASel Table

MuxBSel	MuxBOut
00	ALU's Output
01	Address Register File's Output
10	Memory's Output
11	Instruction Register's (7-0) bits

Table 9: MuxBSel Table

MuxCSel	MuxCOut
0	ALU's (7-0) bits
1	ALU's (15-8) bits

Table 10: MuxCSel Table

Our design can be seen from Figure 6.

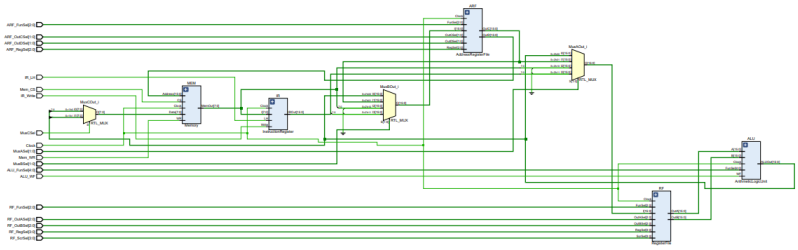

Figure 6: Schematic of Arithmetic Logic Unit System

3 RESULTS [15 points]

3.1 PART - 1

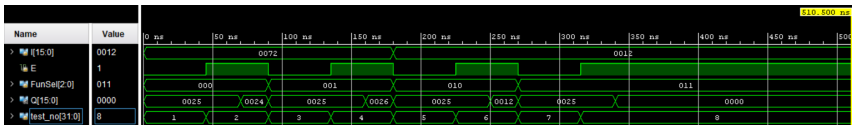

Figure 7: Waveform of 16-bit general purpose register

3.2 PART - 2

3.2.1 PART - 2A

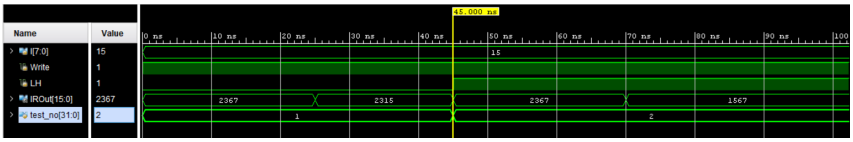

Figure 8: Waveform of 16-bit instruction register

3.2.2 PART - 2B

As no I and RegSel are given at the testcase 1 they appear as red and xxx.

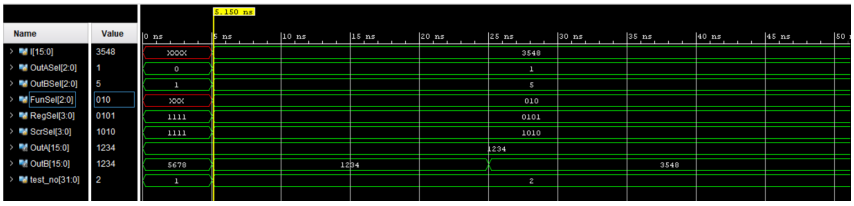

Figure 9: Waveform of register file

3.2.3 PART - 2C

As no FunSel and I are given at the testcase 1 they appear as red and xxx.

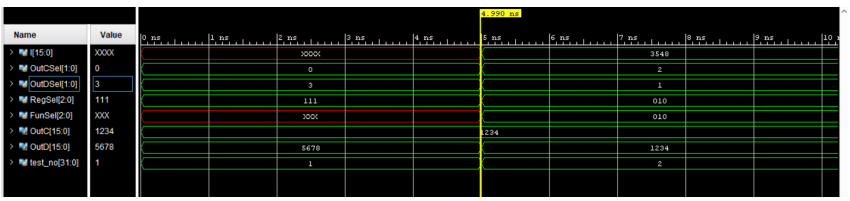

Figure 10: Waveform of address register file

3.3 PART - 3

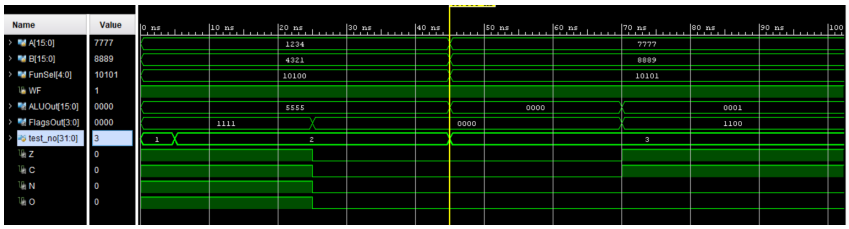

Figure 11: Waveform of Arithmetic Logic Unit

3.4 PART - 4

As no ARF_OutCSel, ARF_OutDSel and IR_LH are given at the testcase 1 they appear as red and xxx.

Figure 12: Waveform of Arithmetic Logic Unit System

4 DISCUSSION [25 points]

In the first part, we built a register module. If (1-bit input) E (enable) is 0, next state of (16-bit output) Q is equal to itself. If enable is 1, according to (3-bit input) FunSel function is selected. According to the selected function next state of output Q is found by register. And these states change according to rising edge of Clock input.

In the second part's a, we built an instruction register module. When (1-bit input) Write is 1, according to chosen (1-bit input) LH we find the next state of (16-bit output) IROut using the give (16-bit input) I and the current state of IROut. And these states change according to rising edge of Clock.

In the second part's b, we built a register file module. In it we defined 8 registers (R1, R2, R3, R4, S1, S2, S3, S4). They have (16-bit input) I and (3-bit input) FunSel (works same with register FunSel) in them. According to (4-bit input) RegSel (register selection) we decided which of the registers (R1, R2, R3, R4) will be enabled. Similarly, using (4-bit input) ScrSel (scratch selection) we decided which of the scratches (S1, S2, S3, S4) will be enabled. According to (3-bit inputs) OutASel and OutBSel, (16-bit outputs) OutA and OutB is found using selected registers' and scratches' outputs. These states change after waiting for some time.

In the second part's c, we built an address register file module. In it defined three registers: PC (program counter), AR (address register), and SP (stack pointer). With a similar RegSel functionality we chose which of the registers are enabled. According to (2-bit input) OutCSel and OutDSel, (16-bit outputs) OutC and OutD is found using registers' outputs. These states change after waiting for some time.

In the third part, we built an arithmetic logic unit (ALU) module. It has (16-bit inputs) A and B. According to (5-bit input) FunSel, (16-bit output) ALUOut is determined. And if (1-bit input) WF is equal to 1, (4-bit output) Z (zero), N (negative), C (carry), and O (overflow) flags (FlagsOut) change state. These outputs (ALUOut and FlagsOut) are controlled by different clock inputs.

In the fourth part, we built an arithmetic logic unit system module. We used the other modules, we designed, instruction register, register file, address register file, and arithmetic logic unit. We also used the given memory module. We defined three multiplexers. 2 of them (4-bit input and 1-bit output) are MuxA and MuxB. Their output (ALUOut, ARFOutC, MemOut, IROut) is determined by (4-bit input) MuxASel and MuxBSel. MuxAOut (16-bit) is an input for register file, and MuxBOut (16-bit) is an input for address register file. Register file's outputs (16-bit) RFOutA and RFOutB are inputs for ALU. ALU's output ALUOut (16-bit) is input for all multiplexers. Third multiplexer MuxC takes input according to MuxCSel (if it is equal 0, MuxCOut = ALUOut[7:0]; if

it is equal to 1, MuxCOut = ALUOut[15:8]). And MuxC's output (8-bit), MuxCOut, is an input for Memory.

5 CONCLUSION [10 points]

During this project, we faced various problems. First of all, this was the first time we encountered with language verilog and the environment we used for compiling it. So it took a little bit of time to get used to it. Later, we had problems when we tried to test our module. Because we did not know we need to be careful about naming the variables according to simulations. And also we think that quantity of tests were insufficient. We know that, it is also part of project to understand what our code should do and what outputs it gives when specific inputs are given. But, for some points we had difficulty at understanding the file instructions and we expected to understand that points from the simulation tests. But, there were not any tests for that part.

In this project, we learned how to design different modules for different parts of CPU and how to combine them. When doing so, we also learned how those parts execute the given functions.