ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E

COMPUTER ORGANIZATION
PROJECT 1 REPORT

CRN : 21334

LECTURER : Gökhan İnce

GROUP MEMBERS:

150210007 : Mustafa Bozdoğan

150210014 : Enes Saçak

SPRING 2024

Contents

1 INTRODUCTION

In the previous project, we designed a simple computer model consisting of a memory unit, arithmetic logic unit (ALU) and many registers used for different purposes, where these are selected and connected.1 with multiplexers, which can store data and perform microoperations on the them. In this project, we designed a central processing unit (CPU) that controls the system we had previously designed by changing the selectors of the multiplexers, enablers in some units, and similar function inputs. The output obtained at the end of this project can be described as a computer, albeit at a simple level.

1.1 Task Distribution

In this project, as in the previous one, we worked together to design and implement the system, control and test the operating status, and prepare the report.

2 MATERIALS AND METHODS

2.1 Terminology for Previous Units

Register File (RF): Consist of 4 (16-bit) general purpose registers (R1, R2, R3, R4) and 4 (16-bit) scratch (S1, S2, S3, S4) registers.

Address Register File (ARF): Consist of 3 (16-bit) registers (AR - PC - SP) used for memory operations (read - write).

Instruction Register (IR): An 16-bit register to store operations and data fetched from memory unit.

Memory: A unit that hold 8-bit values for data and operations.

Arithmetic Logic Unit (ALU): A unit that can perform 8-bit and 16-bit arithmetic and logic operations.

2.2 Instructions

The 16-bit IR is filled according to the information coming from the memory as a result of two clock cycles. These 16 bits have different meanings according to 2 different classifications.

First Case: 3 parts as OPCODE, RSEL, ADDRESS. Classification used in memory-related writing and reading operations.

Second Case: 5 parts as OPCODE, S, DSTREG, SREG1, SREG2. Classification expressing situations other than the first.

OPCODE[15:10]: A 6-bit number that determines which of 34 different operations will be performed.

RSEL[9:8]: A 2-bit number indicating which of the 4 registers in the RF will be used. The value used for Rx in operations is determined by RSEL.

ADDRESS[7:0]: The value that is loaded with the 8-bit value in memory and can be used later in operations related to that address.

S[9:9]: A 1-bit value that determines whether the flags will change for the operations to be performed on the ALU.

DSTREG[8:6]: The destination register indicates whether the values selected or resulting from the operations are in ARF or RF.

SREG1[5:3]: First source register that determines whether the value to be processed or moved is in ARF or RF.

SREG2[2:0]: Second source register that determines whether the value to be processed or moved is in ARF or RF.

2.3 Sequence Counter

We have a register named T that acts as a sequence counter in the system. However, when counting clocks, we look at bit positions instead of looking at T numerically. Therefore, T can hold up to 8 clock cycles, which is more than enough for our operations.

After the sequence counter is reset to 0, we use the first two clock cycle to load instruction register. Filling the instruction register should be done as IR $\leftarrow$ M[PC], but although the instruction register is 16-bit, the memory can give 8-bit output. Therefore, in the first cycle, we fetch data from the specified address on the PC and load the LSB part of the IR, while at the same time incrementing the PC. Thus, in the second cycle, we load the MSB part of the IR from the next line in the memory, but since we do not use the same memory part in the next process, the increment of the PC still remains open and the PC goes to the next clock cycle by increasing. In the end, we fill the instruction register in the first two cycles.

2.4 Tasks Used in Project

We can use these two tasks anywhere that contains SREG1, whether it contains SREG2 or not. For operations with only SREG1 register, we call the GET_SREG1 task,

and if SREG1 is in ARF, it moves it to RF, if it is already in RF, it calls the GET_SREG2 task. GET_SREG2 task sends SREG1 to the ALU either in the first cycle or in the second cycle, depending on the initial situation, and ensures that the process is completed. There are 4 different situations for operations in which both SREG1 and SREG2 are used. If both of them work in RF, in the first clock cycle, GET_SREG1 calls GET_SREG2 without doing anything, and it sends it to the EXECUTE_ALU task without doing anything and the operations are completed. If SREG1 is in RF and SREG2 is in ARF, GET_SREG1 calls GET_SREG2 without doing anything and sends SREG2 to RF. In the other clock cycle, these two values are sent to the ALU and the process is completed. If SREG1 is in ARF and SREG2 is in RF, GET_SREG1 sends SREG1 to RF in the first clock cycle. In the other clock cycle, GET_SREG2 directly calls EXECUTE_ALU and the process is completed. If both are in ARF, GET_SREG1 sends SREG1 to RF in the first clock cycle, and GET_SREG2 sends SREG2 to RF in the other clock cycle. At the last clock cycle, we still call GET_SREG2 and it directly calls the EXECUTE_ALU task and completes the process.

2.4.1 GET_SREG1 and GET_SREG2

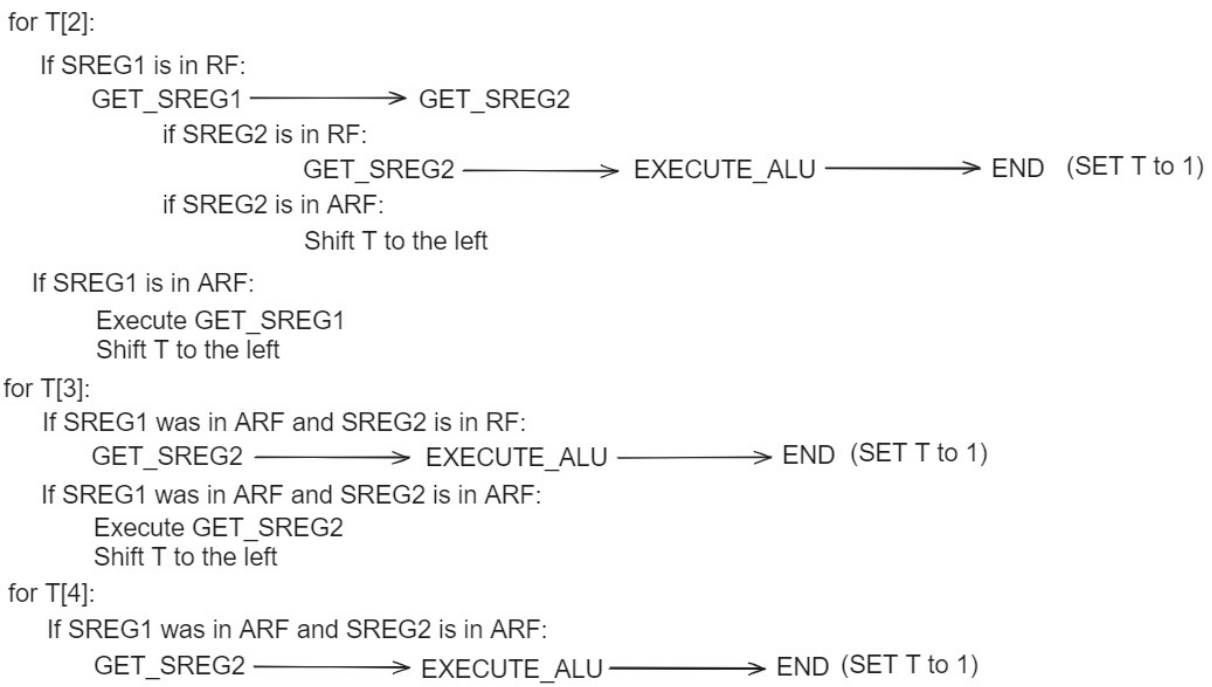

Figure 1: Test

2.4.2 EXECUTE_ALU

In this task, we set ALU_FunSel according to what the OPCODE is, and we set RF_OutASel and RF_OutBSel to get the two operands that the ALU will perform operations on from RF.

2.4.3 END_OPERATION

In this task, we set RF_RegSel, ARF_RegSel, MuxASel, MuxBSel, ARF_FunSel and RF_FunSel depending on where DSTREG is located.

2.5 Operations

In total, we had 34 different operations to complete. Although some of these were different from others, some were very easily grouped. Below, it will be explained which operations are performed in what order and how for each group.

2.5.1 Operation Group 1

BRA (PC $\leftarrow$ PC + VALUE)
BNE (IF Z=0 THEN PC $\leftarrow$ PC + VALUE)
BEQ (IF Z=1 THEN PC $\leftarrow$ PC + VALUE)

The value specified here is the least significant 8-bit of the instruction register that has been loaded before. To add the value to the PC, we need 3 clock cycles, which we will explain below. We used an if condition to make it work for all of these 3 operations.

T[2]: Since we would perform the summation process in the ALU, we had to send the PC to the RF first. We used scratch registers to avoid changing general purpose registers in RF. To perform this operation, we set ARF's OutDSel 00 and made it select PC among the other 3 registers. At the same time, for MuxASel, which has output towards RF, we set our selector to 01 and sent the output from ARF. For RF, we set RF_ScrSel to 0111 and RF_FunSel to 010 and loaded our PC value into the S1 register.

T[3]: In this part, since we needed to send the value from IR to RF, we set the MuxASEl selector to 11. Since we wanted to use S2 in RF, we set our RF_ScrSel to 1011 and left our RF_FunSel to load as before. As a result of these two clock cycles, S1 and S2 were filled with PC and value.

T[4]: Here, we had to send the values in S1 and S2 to the ALU and send the result

to the PC. That's why we made RF_OutASel = 100 and RF_OutBSel = 101 and sent S1 and S2 to the ALU. We made ALU_FunSel 10100 to perform 16-bit addition operation. In order for this output to return to the PC, we set the selector of MuxBSel, which gives the correct output to the ARF, to 00 and enabled it to select the input coming from the ALU. We set ARF_RegSel = 011 and ARF_FunSel = 3'010, loaded the PC and completed the process.

2.5.2 Operation Group 2

POP (SP ← SP + 1, Rx ← M[SP])

What is meant by Rx here is one of the 4 general purpose registers in RF. And as a result of three clock cycles, we had to assign the values at the address specified with M[SP] to one of these registers.

T[2]: In the pop operation, unlike push, we need to do SP ← SP + 1 in the first cycle. We made ARF FunSel = 3'b001, which means increment, and ARF RegSel = 3'b110, so that the SP goes to the next clock cycle by increasing by one.

T[3]: We had to send the value in the second clock cycle M[SP] to the least significant 8-bit of one of the general purpose registers. To select SP, we set ARF_OutDSel = 11. To send the value from here to RF, we set MuxASel = 10 and used the formula ((1000 >> RSEL) XOR 1111) to set enables according to which register in RF would be selected. In this way, the bit specified for the desired register would be 0 and its enable would be on. For RF_FunSel, we set 101, which means least significant 8-bit load.

T[4]: The only thing that is different in this clock cycle from the previous one is that RF_FunSel is 110 because we had to fill the most significant 8-bit of the register specified with Rx here.

2.5.3 Operation Group 3

PSH (M[SP] ← Rx, SP ← SP – 1)

Here, the operation takes 2 clock cycles, unlike the pop operation, because the stack pointer always holds the position of the place to be pushed.

T[2]: Here we use the formula RF_OutASel = 0, RSEL to send the 16-bit data coming from Rx to memory, this determines exactly which register will give output. ALU_FunSel

was set to 10000 to directly transfer the value from here. Since we wanted to use the most significant 8-bit of the value coming from the ALU in this clock cycle, we set MuxCSel = 1. We set the enable inputs to be able to write to the memory, we set ARF_OutDSel = 11 to write to the part of the memory specified by SP. In order for the SP to work with the stack logic and go to the upper part in the next clock cycle, a decrement process was required. For this, ARF_FunSel was set to 000 and ARF_RegSel was set to 3'110.

T[3]: In this clock cycle, unlike the previous one, we only set MuxCSel = 0 and wrote the least significant 8-bit to memory.

2.5.4 Operation Group 4

INC (DSTREG ← SREG1 + 1)
DEC (DSTREG ← SREG1 - 1)
MOVS (DSTREG ← SREG1, Flags will change)

In these operations, we had to send the data in the source register SREG1 to the destination register DSTREG without changing it and then increment or decrement it according to the operation. These operations generally take two clock cycles, except that SREG1 and DSTREG are the same register or the operation is MOVS.

T[2]: If SREG1 and DSTREG are the same, we set ARF_RegSel or RF_RegSel to activate them (actually one), depending on whether these registers are in ARF or RF, and We set ARF_FunSel or RF_FunSel 001 as increment or 000 as decrement depending on the type of operation and we finish the process (if the operation is MOVS we didn't set these). If they are not the same, by looking at the first bit of SREG1, we divide the process into two, depending on whether it is in RF or ARF. If SREG1 is in RF, we need to pass it through the ALU to send it to DESTREG. For this, we use the formula RF_OutASel = {1'b0, SREG1[1:0]} and set ALU_FunSel to 10000 so that the ALU gives output without making any changes. Then we run the t_end task to get the ALU output to DSTREG.

T[3]: Here, we just divide it into cases according to where the DSTREG is, adjust the RegSel and FunSel of the ARF or RF according to whether the operation is increment or decrement, and finish the process.

2.5.5 Operation Group 5

LSL (DSTREG ← LSL SREG1)

LSR (DSTREG ← LSR SREG1)
ASR (DSTREG ← ASR SREG1)
CSL (DSTREG ← CSL SREG1)
CSR (DSTREG ← CSR SREG1)
NOT (DSTREG ← NOT SREG1)

In these operations, we used the t_sreg1, t_sreg2, t_alu and t_end tasks that we explained above.

T[2]: We called t_sreg1 in this clock cycle. If SREG1 is already in RF, t_sreg1 directly calls t_sreg2, which in turn directly calls t_alu, and t_alu sets the ALU_FunSel according to the OPCODE of the operation to be performed, and then it calls t_end within itself and we complete this process in this clock cycle. If SREG1 is not in RF but is in ARF, t_sreg1 adjusts OutCSel according to which register's value will be output from ARF, sets MuxASel to 2'b01 and sets RF_FunSel to 010, allowing the value from ARF to be loaded and waiting for the other clock cycle.

T[3]: Here, we call the sreg_2 task and it sends it directly to the t_alu task for operations where SREG2 is not available. According to the operation to be performed here, ALU_FunSel is set and t_end task is called. In this task, depending on the location of DSTREG, it sets one of MuxASel or MuxBSel to 00, one of ARF_FunSel or RF_FunSel is set to 010, and the load is loaded to the register enabled from ARF_RegSel or RF_RegSel. Here we finish the process.

2.5.6 Operation Group 6

AND (DSTREG ← SREG1 AND SREG2)
ORR (DSTREG ← SREG1 OR SREG2)
XOR (DSTREG ← SREG1 XOR SREG2)
NAND (DSTREG ← SREG1 NAND SREG2)
ADD (DSTREG ← SREG1 + SREG2)
ADC (DSTREG ← SREG1 + SREG2 + CARRY)
SUB (DSTREG ← SREG1 - SREG2)
ADDS (DSTREG ← SREG1 + SREG2, Flags will change)
SUBS (DSTREG ← SREG1 - SREG2, Flags will change)
ANDS (DSTREG ← SREG1 AND SREG2, Flags will change)
ORS (DSTREG ← SREG1 OR SREG2, Flags will change)
XORS (DSTREG ← SREG1 XOR SREG2, Flags will change)

Here we again use the tasks we mentioned before. Depending on the position of SREG1 and SREG2, we determine in which clock cycle the process will end. If you do not know how these tasks work, please see above.

T[2]: In this clock cycle, we call the t_sreg1 task. If the first bits of both SREG1 and SREG2 are 1, which indicates that they are both in the RF, we finish the process directly.

T[3]: In this clock cycle, we call the t_sreg2 task. If only one of the first bits of SREG1 and SREG2 is 1, which indicates that one of them is in RF and other is in ARF, we finish the process directly.

T[4]: In this clock cycle, we call again the t_sreg2 task. If the process continues this far, it means that the first bit of both SREG1 and SREG2 is 0, which means that both source registers are in ARF. Here we finalize the process once and for all.

2.5.7 Operation Group 7

MOVH (DSTREG[15:8] ← IMMEDIATE (8-bit))
MOVL (DSTREG[7:0] ← IMMEDIATE (8-bit))

In these operations, the value specified as IMMEDIATE is the least significant 8-bit of the instruction register that has been loaded before. We need to move the 8-bit part coming out of here to DSTREG.

T[2]: Here, if the first bit of DSTREG is 0, it means that it is in ARF, and we need to send the IMMEDIATE value coming from IR by setting MuxBSel to 11, and we set ARF_RegSel according to which register it will go to. If the first bit is 1, it means it is in RF, and we need to send the IMMEDIATE value to RF by setting MuxAsel to 11, and we choose RF_RegSel accordingly. For both different conditions of both processes, we set ARF_FunSel and RF_FunSel to 101 or 110, which is determined by the difference between DSTREG[15:8] and DSTREG[7:0].

2.5.8 Operation Group 8

LDR(16-bit) (Rx ← M[AR] (AR is 16-bit register))

In this operation, we need to send the value in the part of the memory specified with AR to the Rx register. Since the Rx register is a 16-bit register located in RF and the memory consists of 8-bit parts, we need to complete this in 2 clock cycles.

T[2]: Since we need to use AR in this clock cycle, we set OutDSel to 10 to select AR, we set MuxASel to 10 so that the value coming out of memory goes to Rx, that is, a register in RF, and we determine RF_RegSel according to which register will be enabled. Also, in this clock cycle, we set ARF_RegSel to 110 and ARF_FunSel to 001 so that AR enters the next clock cycle with an increase of 1.

T[3]: Here, we only change RF_FunSel as 110 because this time we need to fill in the most significant 8-bit. Here, we do not change the increment conditions for AR because we think it should move to the next memory section.

2.5.9 Operation Group 9

STR(16-bit) (M[AR] ← Rx (AR is 16-bit register))

Here we need to write the value specified in Rx to the part of the memory specified by AR. The registers in Rx, or RF, are 16-bit, but since the memory is 8-bit, this process takes 2 clock cycles.

T[2]: Since we want to write the least significant 8-bit of the value in RF to memory in this clock cycle, we set RF_OutASel according to which Rx register is, and ALU_FunSel is set to 10000 to direct the value sent by RF directly. Since we will be writing at least significant 8-bit, we set MuxCSel to 0. Also, in this clock cycle, we set ARF_RegSel to 110 and ARF_FunSel to 001 so that AR enters the next clock cycle with an increase of 1.

T[3]: In this clock cycle, we only change MuxCSel to 1, so we need to write the most significant 8-bit of the value in Rx to the part of the memory specified by AR, which was increased in the previous clock. We continue to increase AR in this clock cycle.

2.5.10 Operation Group 10

BX(16-bit) (M[SP] ← PC, PC ← Rx)

In this process, we first write the PC value to the part of the memory specified with SP and then update the value in Rx by writing it to the PC. These processes take 4 clock cycles: sending the PC to the RF, writing the most significant and least significant 8-bit from the RF to memory, and writing the Rx to the PC.

T[2]: Since we need to load the value in the PC to Rx in this clock cycle, we set the OutCSel to 00 to select the PC, and the MuxASel, which has output towards RF, to 01

to select the PC coming from the ARF. To store the value coming from the PC in the first scratch register, we set RF_ScrSel to 0111 and RF_FunSel to 010.

T[3]: In this clock cycle, we set RF_OutAsel to 100 to send the value we previously loaded to the RF to the ALU and turn off the enablers due to the risk of the values inside the RF changing. We set ALU_FunSel to 10000 so that the value we send to the ALU is transferred to memory without changing. Since we are using a stack pointer, we first set MuxCSel to 1 to fill the most significant 8-bit. Since we want to write to the part of the memory specified by SP, we set ARF_OutDSel to 11. Apart from these, since it works with stack logic, SP-1 should be in the next clock cycle. Therefore, we set ARF_RegSel to 110, which will activate SP, and ARF_FunSel to 000, which means decrement.

T[4]: The only thing different here from the previous one is MuxCSel because this time we need to write the least significant 8-bit of the ALU's output.

T[5]: In the last clock cycle, we need to load the value in Rx to the PC. Therefore, RF_OutASel = {1'b0, RSEL}, which determines RF_OutASel according to which Rx is. We use the formula. To transfer the value sent to the ALU from here without changing, we set ALU_FunSel to 10000. Since we want to send the output from here to ARF, we set MuxBSel to 00, ARF_RegSel to 011 and ARF_FunSel to 010 and finish the process.

2.5.11 Operation Group 11

BL(16-bit) (M[SP] $\leftarrow$ PC, PC $\leftarrow$ Rx)

Here we need to load the values from the part of the memory specified with SP to the PC. We need to increase SP by 1 in the first clock cycle because SP holds the free memory space to be pushed. In the next clock cycle, we fill the PC with the value coming from memory. In the other two hours, we do the same thing we did in the first two.

T[2]: Here we set ARF_FunSel to 110 for increment and ARF_RegSel to 110.

T[3]: Here, we set ARF_RegSel to 011 and ARF_FunSel to 101, which means least significant 8-bit load. Thus, the first 8-bit of the PC is loaded.

T[4]: Here we set ARF_FunSel to 110 for increment and ARF_RegSel to 110.

T[5]: Here, we set ARF_RegSel to 011 and ARF_FunSel to 110, which means most

significant 8-bit load. Thus, the last 8-bit of the PC is loaded

2.5.12 Operation Group 12

LDRIM (Rx ← VALUE (VALUE defined in ADDRESS bits))

Here, the value specified by value is the least significant 8-bit of the IR. We need to load the value here into the register specified by Rx and located in RF.

T[2]: Here we set MuxAsel, which has an output to RF, to 10 to send the VALUE coming from IR. We set RF_FunSel to 010, which means load, and set RF_RegSel to choose which register it is.

2.5.13 Operation Group 13

STRIM (M[AR+OFFSET] ← Rx (AR is 16-bit register) (OFFSET defined in ADDRESS bits))

Here the entire process takes 5 clock cycles in total. The value specified by OFFSET is the least significant 8-bit of IR. In the first clock cycle, we fill S1 with AR, in the second, we fill S2 with OFFSET, in the 3rd clock cycle, we write the result of the operation to AR, and in the other two clock cycles, we load the value from Rx into the memory section specified with AR.

T[2]: Here, to load S1 with AR, we set ARF_OutCSel to 2'b10, MuxASel to 2'b01, RF_FunSel to 010 and RF_ScrSel to 0111.

T[3]: Here, RF_ScrSel is 1011 because we want to fill it to S2, and MuxAsel is 10 because it comes from OFFSET IR.

T[4]: Here we set RF_OutASel and RF_OutBSel to select S1 and S2 and set ALU_FunSel to 10100 to add these two values. To write the output from the ALU to AR, we set MuxBSel to 00, ARF_RegSel to 101 and ARF_FunSel to 010.

T[5]: Here, we set OutASel according to which register is specified in Rx and set it to 10000 to transmit ALU_FunSel without making any changes. Since the value coming from here is at least significant 8-bit, we set MuxCSel to 0.

T[6]: Here, unlike the previous clock cycle, we just set MuxCSel to 1 and finish the operation.

3 RESULTS

3.0.1 EXCEL TESTS

OPCODE (HEX)	SYMBOL	DESCRIPTION	INSTRUCTION REGISTER	RSEL	DSTREG	SREG1	SREG2	ADDRESS	S	IR(HEX)
0x00	BRA	PC ← PC + VALUE	0b000000 00 00101000	R1	-	-	-	0b00101000	-	0x0028
0x01	BNE	IF Z=0 THEN PC ← PC + VALUE	0b000001 00 00101000	R1	-	-	-	0b00101000	-	0x0428
0x02	BEQ	IF Z=1 THEN PC ← PC + VALUE	0b000010 00 00101000	R1	-	-	-	0b00101000	-	0x0828
0x03	POP	SP ← SP + 1, Rx ← M[SP]	0b000011 00 11111111	R1	-	-	-	0b11111111	-	0x0CFF
0x04	PSH	M[SP] ← Rx, SP ← SP – 1	0b000100 10 11111111	R3	-	-	-	0b11111111	-	0x12FF
0x05	INC	DSTREG ← SREG1 + 1	0b000101 0 101 100 100	-	R2	R1	R1	-	0	0x1564
0x06	DEC	DSTREG ← SREG1 – 1	0b000110 0 101 100 100	-	R2	R1	R1	-	0	0x1964
0x07	LSL	DSTREG ← LSL SREG1	0b000111 0 111 110 110	-	R4	R3	R3	-	0	0x1DF6
0x08	LSR	DSTREG ← LSR SREG1	0b001000 0 111 110 110	-	R4	R3	R3	-	0	0x21F6
0x09	ASR	DSTREG ← ASR SREG1	0b001001 0 111 110 110	-	R4	R3	R3	-	0	0x25F6
0x0A	CSL	DSTREG ← CSL SREG1	0b001010 0 111 110 110	-	R4	R3	R3	-	0	0x29F6
0x0B	CSR	DSTREG ← CSR SREG1	0b001011 0 111 110 110	-	R4	R3	R3	-	0	0x3DF6
0x0C	AND	DSTREG ← SREG1 AND SREG2	0b001100 0 011 101 000	-	AR	R2	PC	-	0	0x30E8
0x0D	ORR	DSTREG ← SREG1 OR SREG2	0b001101 0 010 011 111	-	SP	AR	R4	-	0	0x349F
0x0E	NOT	DSTREG ← NOT SREG1	0b001110 0 001 000 100	-	PC	PC	R1	-	0	0x3844
0x0F	XOR	DSTREG ← SREG1 XOR SREG2	0b001111 0 110 100 101	-	R3	R1	R2	-	0	0x3DA5
0x10	NAND	DSTREG ← SREG1 NAND SREG2	0b010000 0 101 011 100	-	R2	AR	R1	-	0	0x415C

Figure 2: Test

3.0.2 EXCEL TESTS

0x11	MOVH	DSTREG[15:8] ← IMMEDIATE (8-bit)	0b010001 0 100 001 010	-	R1	PC	SP	0b10101010	0	0x450A
0x12	LDR	(16-bit) Rx ← M[AR] (AR is 16-bit register)	0b010010 01 11111111	R2	-	-	-	0b11111111	-	0x49FF
0x13	STR	(16-bit) M[AR] ← Rx (AR is 16-bit register)	0b010011 10 11111111	R3	-	-	-	0b11111111	-	0x4EFF
0x14	MOVL	DSTREG[7:0] ← IMMEDIATE (8-bit)	0b010100 0 100 001 010		R1	PC	SP	0b00100100	0	0x510A
0x15	ADD	DSTREG ← SREG1 + SREG2	0b010101 0 011 010 001	-	AR	SP	PC	-	0	0x54D1
0x16	ADC	DSTREG ← SREG1 + SREG2 + CARRY	0b010110 0 101 011 100	-	R2	AR	R1	-	0	0x595C
0x17	SUB	DSTREG ← SREG1 - SREG2	0b010111 0 110 010 001	-	R3	SP	PC	-	0	0x5D91
0x18	MOVS	DSTREG ← SREG1, Flags will change	0b011000 1 010 010 111	-	SP	SP	R4	-	1	0x6297
0x19	ADDS	DSTREG ← SREG1 + SREG2, Flags will change	0b011001 1 011 010 001	-	AR	SP	PC	-	1	0x66D1
0x1A	SUBS	DSTREG ← SREG1 - SREG2, Flags will change	0b011010 1 100 100 100	-	R1	R1	R1	-	1	0x6B24
0x1B	ANDS	DSTREG ← SREG1 AND SREG2, Flags will change	0b011011 1 000 100 001	-	PC	R1	PC	-	1	0x6E21
0x1C	ORRS	DSTREG ← SREG1 OR SREG2, Flags will change	0b011100 0 111 100 100	-	R4	R1	R1	-	0	0x71E4
0x1D	XORS	DSTREG ← SREG1 XOR SREG2, Flags will change	0b011101 1 100 001 010	-	R1	PC	SP	-	1	0x770A
0x1E	BX	M[SP] ← PC, PC ← Rx	0b011110 01 11111111	R2	-	-	-	0b11111111	-	0x79FF
0x1F	BL	PC ← M[SP]	0b011111 11 11111111	R4	-	-	-	0b11111111	-	0x7FFF
0x20	LDRIM	Rx ← VALUE (VALUE defined in ADDRESS bits)	0b100000 10 11111111	R3	-	-	-	0b11111111	-	0x82FF
0x21	STRIM	M[AR+OFFSET] ← Rx (AR is 16-bit register)	0b100001 10 00000111	R3	-	-	-	0b00000111	-	0x8607

Figure 3: Test

3.0.3 BRA TESTS

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:     0, R2:     0, 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 OPCODE: xx
SREG1: xxx SREG2: xxx DSTREG: xxx RSEL: xx, rst: 0


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:     0, R2:     0, 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 OPCODE: xx
SREG1: 101 SREG2: 000 DSTREG: x00 RSEL: xx, rst: 0

Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :    40
Register File Registers: R1:     0, R2:     0, 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 OPCODE: 00
SREG1: 101 SREG2: 000 DSTREG: 000 RSEL: 00, rst: 0
```

Figure 4: Test

3.0.4 BRA TESTS

```
Output Values:
T:   8
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :    40
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     2, S2:     0, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:     x OPCODE: 00
SREG1: 101 SREG2: 000 DSTREG: 000 RSEL: 00, rst: 0


Output Values:
T:  16
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :    40
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     2, S2:    40, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    42 OPCODE: 00
SREG1: 101 SREG2: 000 DSTREG: 000 RSEL: 00, rst: 1


Output Values:
T:   1
Address Register File: PC:    42, AR:     x, SP:     x
Instruction Register :    40
Register File Registers: R1:     0, R2:     0, R3:     0, R4:     0
Register File Scratch Registers: S1:     2, S2:    40, S3:     0, S4:     0
ALU Flags: Z: x, N: x, C: x, O: x
ALU Result: ALUOut:    42 OPCODE: 00
SREG1: 101 SREG2: 000 DSTREG: 000 RSEL: 00, rst: 0
```

Figure 5: Test

3.0.5 INC TESTS

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:    10, R2:     0, 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 OPCODE: xx
SREG1: xxx SREG2: xxx DSTREG: xxx RSEL: xx, rst: 0


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:    10, R2:     0, 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 OPCODE: xx
SREG1: 100 SREG2: 100 DSTREG: x01 RSEL: xx, rst: 0


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :  5476
Register File Registers: R1:    10, R2:     0, 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:    10 OPCODE: 05
SREG1: 100 SREG2: 100 DSTREG: 101 RSEL: 01, rst: 0
```

Figure 6: Test

3.0.6 INC TESTS

```
Output Values:
T:   8
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :  5476
Register File Registers: R1:    10, R2:    10, 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:    10 OPCODE: 05
SREG1: 100 SREG2: 100 DSTREG: 101 RSEL: 01, rst: 1


Output Values:
T:   1
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :  5476
Register File Registers: R1:    10, R2:    11, 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:    10 OPCODE: 05
SREG1: 100 SREG2: 100 DSTREG: 101 RSEL: 01, rst: 0
```

Figure 7: Test

3.0.7 DEC TESTS

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:    10, R2:     0, 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 OPCODE: xx
SREG1: xxx SREG2: xxx DSTREG: xxx RSEL: xx, rst: 0


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:    10, R2:     0, 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 OPCODE: xx
SREG1: 100 SREG2: 100 DSTREG: x01 RSEL: xx, rst: 0


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :  6500
Register File Registers: R1:    10, R2:     0, 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:    10 OPCODE: 06
SREG1: 100 SREG2: 100 DSTREG: 101 RSEL: 01, rst: 0
```

Figure 8: Test

3.0.8 DEC TESTS

```
Output Values:
T:   8
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :  6500
Register File Registers: R1:    10, R2:    10, 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:    10 OPCODE: 06
SREG1: 100 SREG2: 100 DSTREG: 101 RSEL: 01, rst: 1


Output Values:
T:   1
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register :  6500
Register File Registers: R1:    10, R2:     9, 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:    10 OPCODE: 06
SREG1: 100 SREG2: 100 DSTREG: 101 RSEL: 01, rst: 0
```

Figure 9: Test

3.0.9 MOVH TESTS

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:    10, R2:     0, 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 OPCODE: xx
SREG1: xxx SREG2: xxx DSTREG: xxx RSEL: xx, rst: 0


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:    10, R2:     0, 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 OPCODE: xx
SREG1: 001 SREG2: 010 DSTREG: x00 RSEL: xx, rst: 0


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 17674
Register File Registers: R1:    10, R2:     0, 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 OPCODE: 11
SREG1: 001 SREG2: 010 DSTREG: 100 RSEL: 01, rst: 1


Output Values:
T:   1
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 17674
Register File Registers: R1:  2570, R2:     0, 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 OPCODE: 11
SREG1: 001 SREG2: 010 DSTREG: 100 RSEL: 01, rst: 1
```

Figure 10: Test

3.0.10 MOVL TESTS

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:    10, R2:     0, 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 OPCODE: xx
SREG1: xxx SREG2: xxx DSTREG: xxx RSEL: xx, rst: 0


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:    10, R2:     0, 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 OPCODE: xx
SREG1: 001 SREG2: 010 DSTREG: x00 RSEL: xx, rst: 0


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 20746
Register File Registers: R1:    10, R2:     0, 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 OPCODE: 14
SREG1: 001 SREG2: 010 DSTREG: 100 RSEL: 01, rst: 1


Output Values:
T:   1
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 20746
Register File Registers: R1:    10, R2:     0, 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 OPCODE: 14
SREG1: 001 SREG2: 010 DSTREG: 100 RSEL: 01, rst: 1
```

Figure 11: Test

3.0.11 XOR TESTS

```
Output Values:
T:   1
Address Register File: PC:     0, AR:     x, SP:     x
Instruction Register :     x
Register File Registers: R1:     9, R2:     6, 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 OPCODE: xx
SREG1: xxx SREG2: xxx DSTREG: xxx RSEL: xx, rst: 0


Output Values:
T:   2
Address Register File: PC:     1, AR:     x, SP:     x
Instruction Register :     X
Register File Registers: R1:     9, R2:     6, 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 OPCODE: xx
SREG1: 100 SREG2: 101 DSTREG: x10 RSEL: xx, rst: 0


Output Values:
T:   4
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 15781
Register File Registers: R1:     9, R2:     6, 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:    15 OPCODE: 0f
SREG1: 100 SREG2: 101 DSTREG: 110 RSEL: 01, rst: 1


Output Values:
T:   1
Address Register File: PC:     2, AR:     x, SP:     x
Instruction Register : 15781
Register File Registers: R1:     9, R2:     6, R3:    15, 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:    15 OPCODE: 0f
SREG1: 100 SREG2: 101 DSTREG: 110 RSEL: 01, rst: 0
```

Figure 12: Test

4 DISCUSSION

We also tested our implementation with example outputs we get the same results for the most important part (R1, R2, R3, R4, SP, AR, PC) the other values can be differ from code to code since they all about implementation and there is not only one way to solve any given operation also we used our own tests for each of the given operations which ended up giving the correct results for these cases there were some mistakes while

running the output since we wrote all our tests in one "Ram.mem". Therefore after some writing operations we were overriding some expected test values and changing them by something else. This was the reason we were giving wrong outputs by some cases but if we were to test our outputs and test cases one by one we were getting the correct results as i mentioned.

5 CONCLUSION

Here is our design:

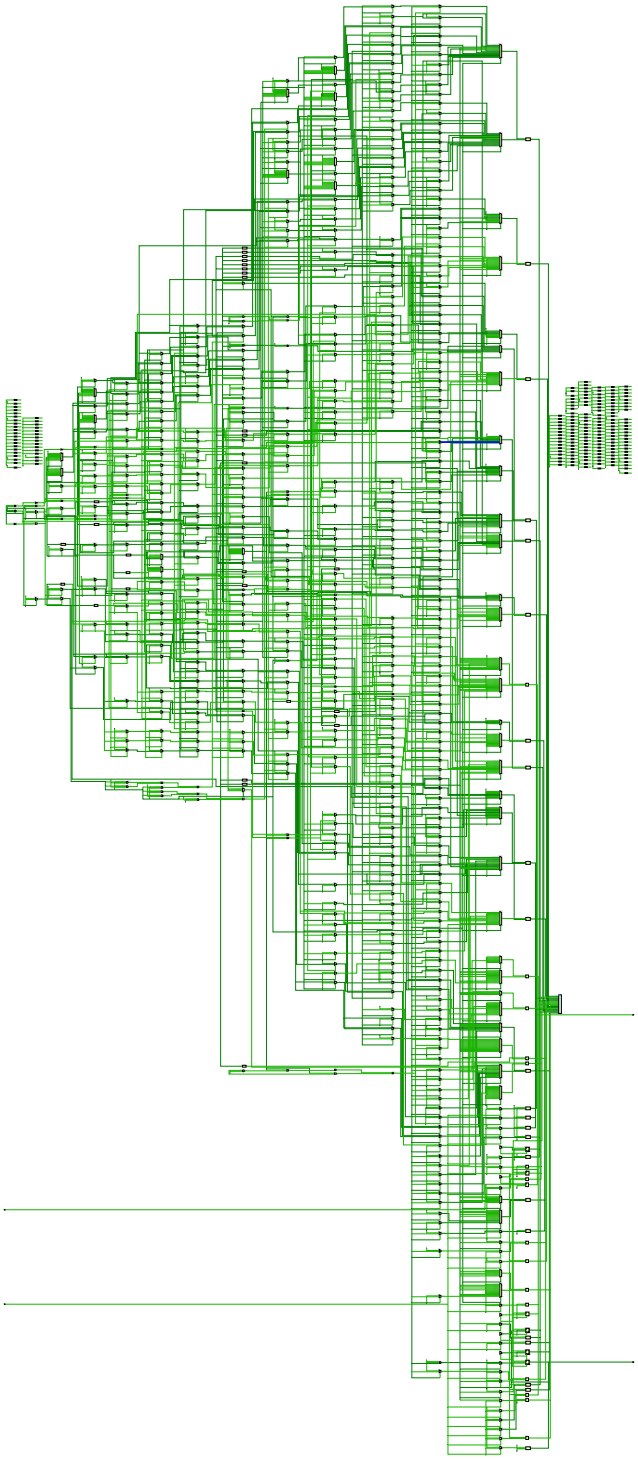

Figure 13: Elaborated Design

We did not experience much difficulty in this project because we were experienced thanks to the previous one. But again, some things in the pdf containing the project instructions were not understandable at all. To give an example, we could not decide whether the flags in the ALU would change according to the S bit or only in the parts specified in the project PDF. Additionally, the fact that we always use the same region for OFFSET, VALUE and IMMEDIATE raised questions in our minds. However, thanks to the help of our assistants, we solved these problems. We spent a long time adjusting the clock timings in this project because we were getting unexpected results that contradicted our logic. These may be because we do not fully understand the working principle of Verilog and have not received detailed training on this subject.

REFERENCES