ISTANBUL TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

BLG 222E
COMPUTER ORGANIZATION REPORT

PROJECT NO	:	3
SUBMISSION DATE	:	15.05.2019
GROUP NO	:	12

GROUP MEMBERS:

150160151	:	Esin Ece AYDIN
150170016	:	Umit BAŞAK
150180704	:	Cihat AKKİRAZ
150180705	:	Batuhan Faik DERİNBAY
150180707	:	Fatih ALTINPINAR

SPRING 2019

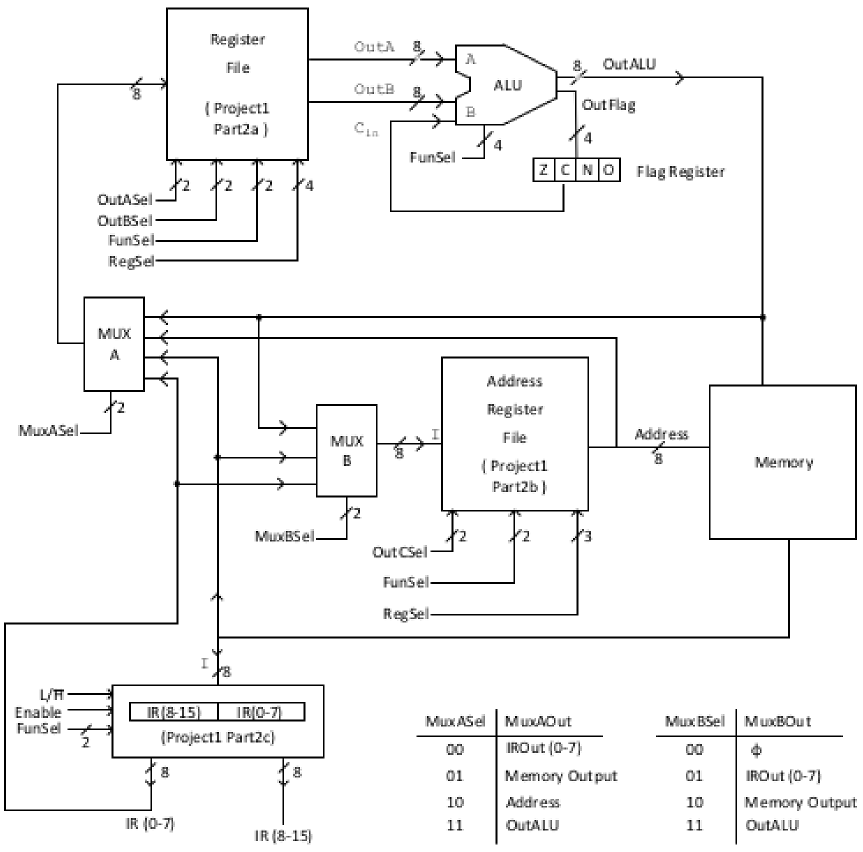
1 INTRODUCTION

Control unit which is a part of central processing unit (CPU) generates timing and control signals for the operations of the computer. The control unit communicates with the arithmetic logic unit (ALU), and main memory. It controls the communication between processor, memory and the various peripherals. It also informs the ALU of the operation to be performed on data.

Implementation of control unit can be done by two ways: combinational logic circuits (Hard-wired), and microprogram. In this project hard-wire control unit of basic computer which operates several functions is designed. This design is based on the organization in Project-2.

2 METHODOLOGY

2.1 The Organization



MuxASel	MuxAOut
00	IROut (0-7)
01	Memory Output
10	Address
11	OutALU

MuxBSel	MuxBOut
00	ϕ
01	IROut (0-7)
10	Memory Output
11	OutALU

Figure 1: The organization that is given in Project-2

The control unit is designed considering organization that is given in the Project 2. To perform particular operation there are three stages: Fetch stage, Decode stage and Execute stage.

2.2 Sequence Counter

To deal with complexity of the controller, sequence counter method is used. Sequence counter determines and controls each sequential next state. After execution of particular instruction, sequence counter is set 0.

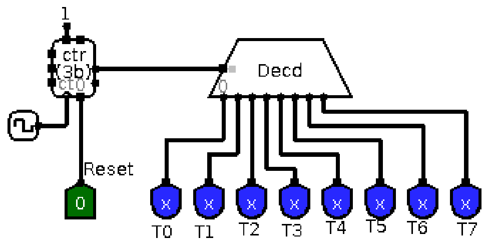


Figure 2: Sequence Counter

2.3 Fetch & Decode

The first step of performing particular instruction is fetching and decoding it. Program instruction is pulled from memory and stored in main memory. Program Counter(PC) keeps the address location of the instruction. Program counter and instructions are loaded in the Instruction Register(IR). Then, Program Counter is incremented by 1.

Decoder interprets the opcode/function of the instruction within the IR. In this architecture there are two types instructions. Decoding occurs for two types differently due to their format.

(1)Instructions with an address reference has format as given below

- The OPCODE is a 5-bit field (See Table 1 for the definition).
- The REGSEL is a 2-bit field (See left side of Table 2 for the definition).
- The ADDRESSING MODE is a 1-bit field (See Table 3 for the definition).
- The ADDRESS is 8 bits

OPCODE	REGSEL	ADDRESSING MODE	ADDRESS

Table 1: Instructions with an address reference

The fact that this type of instruction is a memory instruction, the execution phase will be during the next clock cycle. If the instruction has an indirect address, the effective address is read from main memory, and any required data is fetched from main memory to be processed and then placed into data registers (clock cycle: T3). If the instruction is direct, nothing is done during this clock cycle. If this is an I/O instruction or a register instruction, the operation is performed during the clock cycle.

(2)Instructions without address reference has format as given below

- The OPCODE is a 5-bit field (See Table 3 for the definition).
- DESTREG is a 3-bit field which specifies the destination register (See the right side of Table 4 for the definition).
- SRCREG1 is a 3-bit field which specifies the first source register (See the right side of Table 4 for the definition).
- SRCREG2 is a 3-bit field which specifies the second source register (See the right side of Table 4 for the definition).

OPCODE	DESTREG	SRCREG1	SRCREG2

Table 2: Instructions without address reference

The circuit to fetch an instruction and decode it is implemented as given figure below. Each

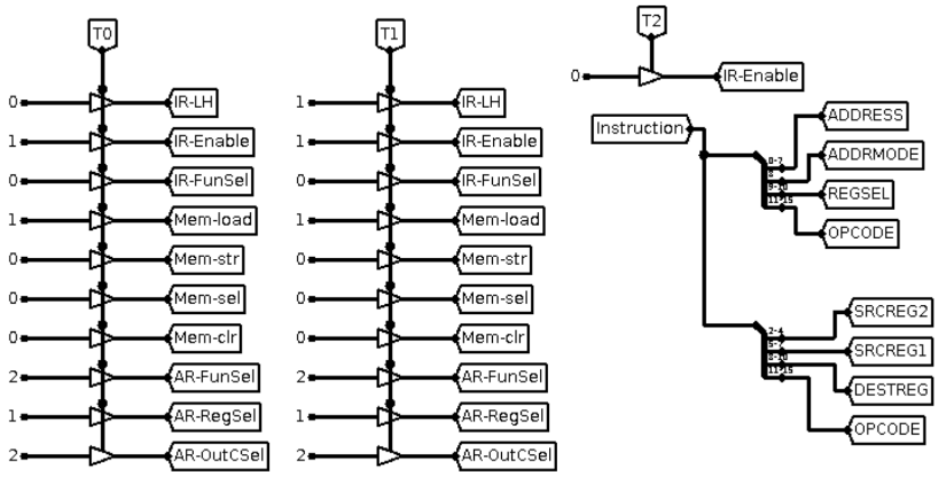


Figure 3: Fetch & Decode

2.4 Operations

In this architecture, 15 operations that are given in the table below can be executed.

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

Table 3: OPCODE field and SYMBols for operations and their descriptions

REGSEL	REGISTER
00	R3
01	R2
10	R1
11	R0

DESTREG/SRCREG1/SRCREG2	REGISTER B
000	R3
001	R2
010	R1
011	R0
100	AR
101	SP
110	PC
111	PC

Table 4: REGSEL(left) and DESTREG/SRCREG1/SRCREG2 (Right) select the register of interest for a particular instruction

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

Table 5: Addressing modes

2.4.1 ADD OPCODE: 0x00

The very first operation that this design can execute is an addition operation (See Table 3). For the addition operation because the instruction is not an address reference it is decoded as shown in Table 2. Since there are many similar non address reference operations that our architecture can execute, analogous circuits with very little differences are used for alike operations such as SUB, OR, AND etc. Therefore a detailed explanation of the design seen in Figure 4 will be given under this subsection. Afterwards, only the small differences of alike operations will be mentioned.

To briefly explain the structure of control circuit used for operations ADD, SUB and alike, executed procedures has to be examined (See Figure 4).

- T2
  - During the third timing cycle of the architecture (The first timing cycle for the operation), the design checks whether the SRCREG1 is from the Address Register File (ARF) block or not.
  - If the SRCREG1 is indeed in the ARF block then its content has to be copied over to the Register File (RF) block in order to be operated by the Arithmetic Logic Unit (ALU). For that a temporary register needs to be chosen from the RF block.
  - The temporary register that the design selects depends on the SRCREG2. If R0 is the second operand (if SRCREG2 = $0x3$), then in T2, contents of SRCREG1 is copied in to R1. Otherwise, where R0 is not the second operand, since contents of the R0 is not important, data stored in SRCREG1 is copied in to R0.
  - If SRCREG1 is not from the ARF block, in this timing cycle the circuit doesn't perform any operations.
- T3
  - During the forth timing cycle of the architecture (The second timing cycle for the operation), the design checks whether the SRCREG2 is from the Address Register File (ARF) block or not.

  - If the SRCREG2 is indeed in the ARF block then its content has to be copied over to the Register File (RF) block in order to be operated by the ALU. For that a temporary register needs to be chosen from the RF block.
  - The temporary register that the design selects depends on the SRCREG1. If R3 is the first operand (if SRCREG1 = $0x1$), then in T3, contents of SRCREG2 is copied in to R2. Otherwise, where R3 is not the first operand, since contents of the R3 is not important, data stored in SRCREG2 is copied in to R3.
  - If SRCREG2 is not from the ARF block, in this timing cycle the circuit doesn't perform any operations.

- T4
  - During the fifth timing cycle of the architecture (The third timing cycle for the operation), the design performs the requested -in this case addition- operation. Depending on whether SRCREG1 and/or SRCREG2 is from the ARF block or not, operand registers of the ALU differ. Because of that, the design needs to decide the input lines (operands) of the ALU.
  - There are 6 possible ways that source registers can be given, hence input lines of the ALU can be chosen.
    1. If neither SRCREG1 nor SRCREG2 is from the ARF block, it means no data needs to be copied over to the RF block. Therefore input lines of the ALU is directly determined by the two least significant bits of SRCREGX lines.
    2. If only SRCREG1 is from the ARF block and SRCREG2 is not R0. Then first operand of the ALU is chosen as R0 and the second operand is determined by the two least significant bits of SRCREG2.
    3. If only SRCREG1 is from the ARF block and SRCREG2 is R0. Then first operand of the ALU is chosen as R1 and the second operand is determined by the two least significant bits of SRCREG2.
    4. If only SRCREG2 is from the ARF block and SRCREG1 is not R3. Then second operand of the ALU is chosen as R3 and the first operand is determined by the two least significant bits of SRCREG1.
    5. If only SRCREG2 is from the ARF block and SRCREG1 is R3. Then second operand of the ALU is chosen as R2 and the first operand is determined by the two least significant bits of SRCREG1.
    6. If both of the SRCREG1 and SRCREG2 is from the ARF block. Then first operand is chosen as R0 and second operand is chosen as R3.

- After selecting the input lines of the ALU, the operation that is needed to be performed by the ALU is chosen with the ALU-FunSel signal. For the addition operation, this signal is determined to be $0x5$.
- At the end, result of the operation (Data on the output line of the ALU) is written to the DESTREG, which is activated by giving LOAD signal to FunSel of the relevant register file block and choosing the necessary register using the RegSel signal. This is achieved via a 3x8 decoder.

- T5
  - Sequence counter is reset.

It should also be mentioned that for all non address reference instructions, inputs of the memory are regulated restrictively so no unwanted changes can be performed on the memory. By implementing all the aforementioned design choices in circuitry, the addition operation is successfully applied to the architecture.

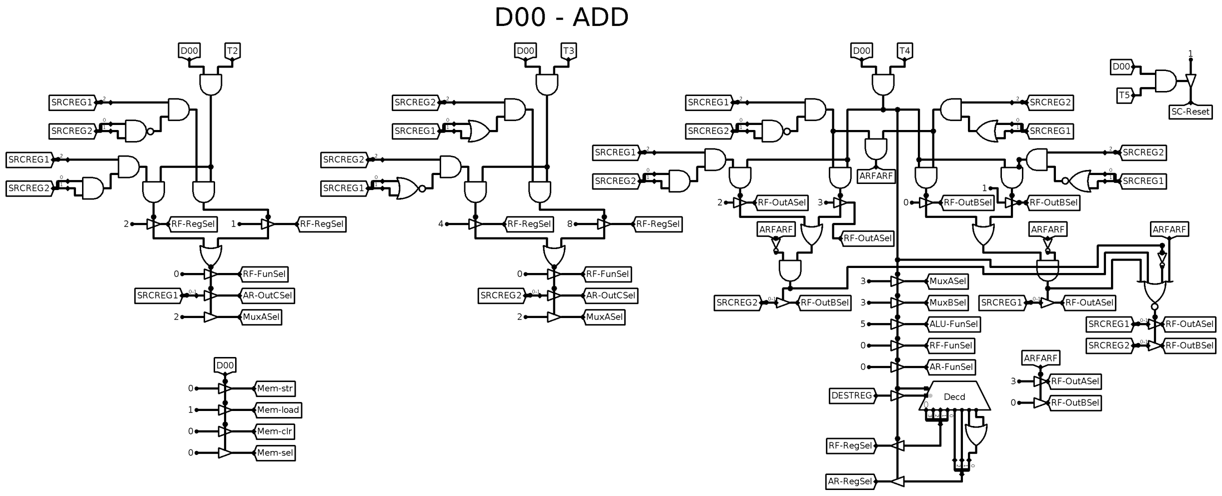


Figure 4: Addition Operation (ADD) Control Circuit

RTL description of the addition operation is given below (See Table 6).

$T2 \cdot SRCREG1(2) \cdot \overline{(SRCREG2(0) \cdot SRCREG2(1))}$	$:R0 \leftarrow SRCREG1$
$T2 \cdot SRCREG1(2) \cdot$ (SRCREG2(0)·$SRCREG2(1)$)	$:R1 \leftarrow SRCREG1$
$T3 \cdot SRCREG2(2) \cdot$ (SRCREG1(0)+SRCREG1(1))	$:R3 \leftarrow SRCREG2$
$T3 \cdot SRCREG2(2) \cdot \overline{(SRCREG1(0) + SRCREG1(1))}$	$:R2 \leftarrow SRCREG2$
$T4 \cdot SRCREG1(2) \cdot \overline{(SRCREG2(0) \cdot SRCREG2(1))}$	$:DESTREG \leftarrow R0 + SRCREG2$
$T4 \cdot SRCREG1(2) \cdot$ (SRCREG2(0)·$SRCREG2(1)$)	$:DESTREG \leftarrow R1 + SRCREG2$
$T4 \cdot SRCREG2(2) \cdot$ (SRCREG1(0)+SRCREG1(1))	$:DESTREG \leftarrow SRCREG1 + R3$
$T4 \cdot SRCREG2(2) \cdot \overline{(SRCREG1(0) + SRCREG1(1))}$	$:DESTREG \leftarrow SRCREG1 + R2$
$T4 \cdot \overline{(SRCREG1(2) \cdot SRCREG2(2))}$	$:DESTREG \leftarrow SRCREG1 + SRCREG2$
$T4 \cdot (SRCREG1(2) \cdot SRCREG2(2))$	$:DESTREG \leftarrow R0 + R3$

Table 6: RTL Description of Addition Operation

2.4.2 SUB OPCODE: 0x01

The second operation that this design can execute is a subtraction operation (See Table 3). For the subtraction operation because the instruction is not an address reference it is decoded as shown in Table 2. Since this operation is very similar to the addition operation, the same structure with very little differences is used (See Figure 5).

The biggest difference between every alike operation is the function selection signal of the ALU. In the case of subtraction operation, ALU-FunSel signal is determined to be $0x6$. Also notice that result needs to be difference between SRCREG2 and SRCREG1 ($RESULT = SRCREG2 - SRCREG1$). Therefore the inputs of the ALU need to be altered. This is achieved by switching the signals of OutASel and OutBSel in the design (See Figure 5).

With that the subtraction operation is successfully implemented.

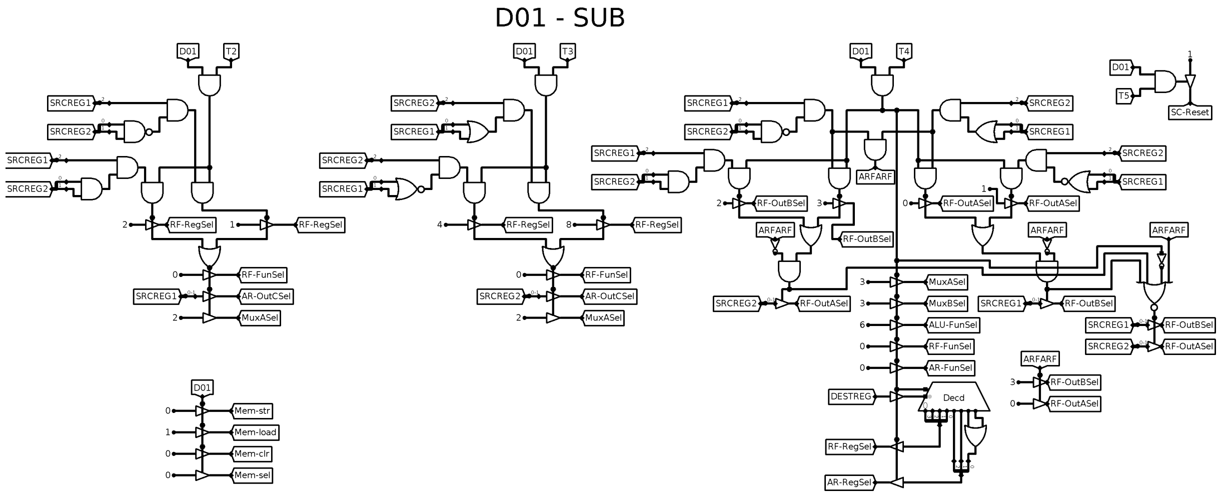


Figure 5: Subtraction Operation (SUB) Control Circuit

RTL description of the subtraction operation is given below (See Table 7).

$T2 \cdot SRCREG1(2) \cdot \overline{(SRCREG2(0) \cdot SRCREG2(1))}$	$:R0 \leftarrow SRCREG1$
$T2 \cdot SRCREG1(2) \cdot$ (SRCREG2(0)$\cdot SRCREG2(1)$)	$:R1 \leftarrow SRCREG1$
$T3 \cdot SRCREG2(2) \cdot$ (SRCREG1(0)+SRCREG1(1))	$:R3 \leftarrow SRCREG2$
$T3 \cdot SRCREG2(2) \cdot \overline{(SRCREG1(0) + SRCREG1(1))}$	$:R2 \leftarrow SRCREG2$
$T4 \cdot SRCREG1(2) \cdot \overline{(SRCREG2(0) \cdot SRCREG2(1))}$	$:DESTREG \leftarrow SRCREG2 - R0$
$T4 \cdot SRCREG1(2) \cdot$ (SRCREG2(0)$\cdot SRCREG2(1)$)	$:DESTREG \leftarrow SRCREG2 - R1$
$T4 \cdot SRCREG2(2) \cdot$ (SRCREG1(0)+SRCREG1(1))	$:DESTREG \leftarrow R3 - SRCREG1$
$T4 \cdot SRCREG2(2) \cdot \overline{(SRCREG1(0) + SRCREG1(1))}$	$:DESTREG \leftarrow R2 - SRCREG1$
$T4 \cdot \overline{(SRCREG1(2) \cdot SRCREG2(2))}$	$:DESTREG \leftarrow SRCREG2 - SRCREG1$
$T4 \cdot (SRCREG1(2) \cdot SRCREG2(2))$	$:DESTREG \leftarrow R3 - R0$

Table 7: RTL Description of Subtraction Operation

2.4.3 DEC OPCODE: 0x02

In order to obtain a decremented value on the register that is given by DESTREG, first the data is moved to destination register, then decremented if DESTREG and SRCREG1 are not equal. If they are equal however, moving the data is not necessary thus allows decrement of the value with one clock cycle. Taking these points to consideration circuit in Figure 7 is built to sustain decrement operation of organization.

If an register's value is decreased, the data is forced to pass from the ALU in order to update flags.

D02 - DEC

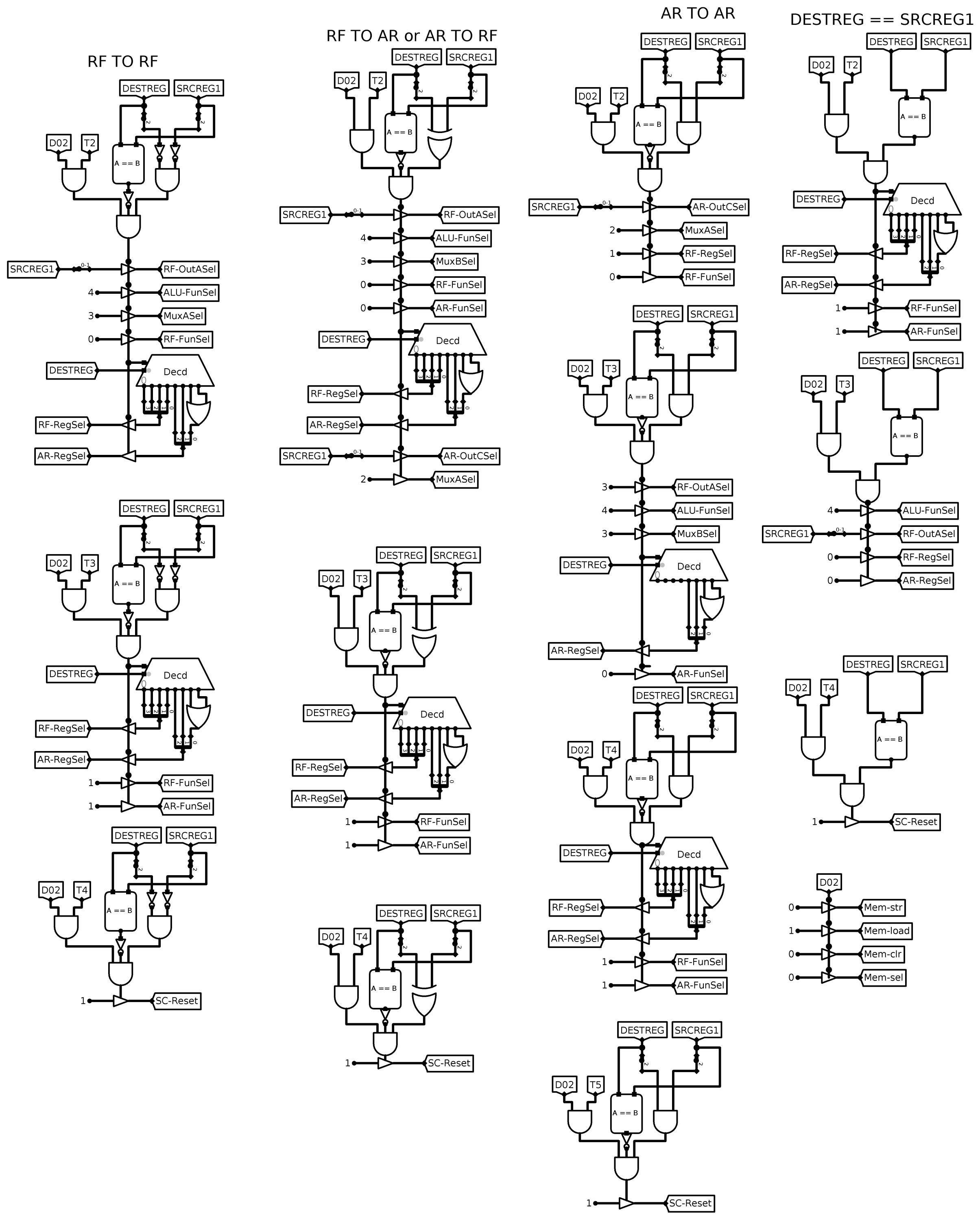


Figure 6: Decrement Operation (DEC) Control Circuit

2.4.4 INC OPCODE: 0x03

Increment function's control signals are completely same but the FunSel of the register file which is set to increment this time.

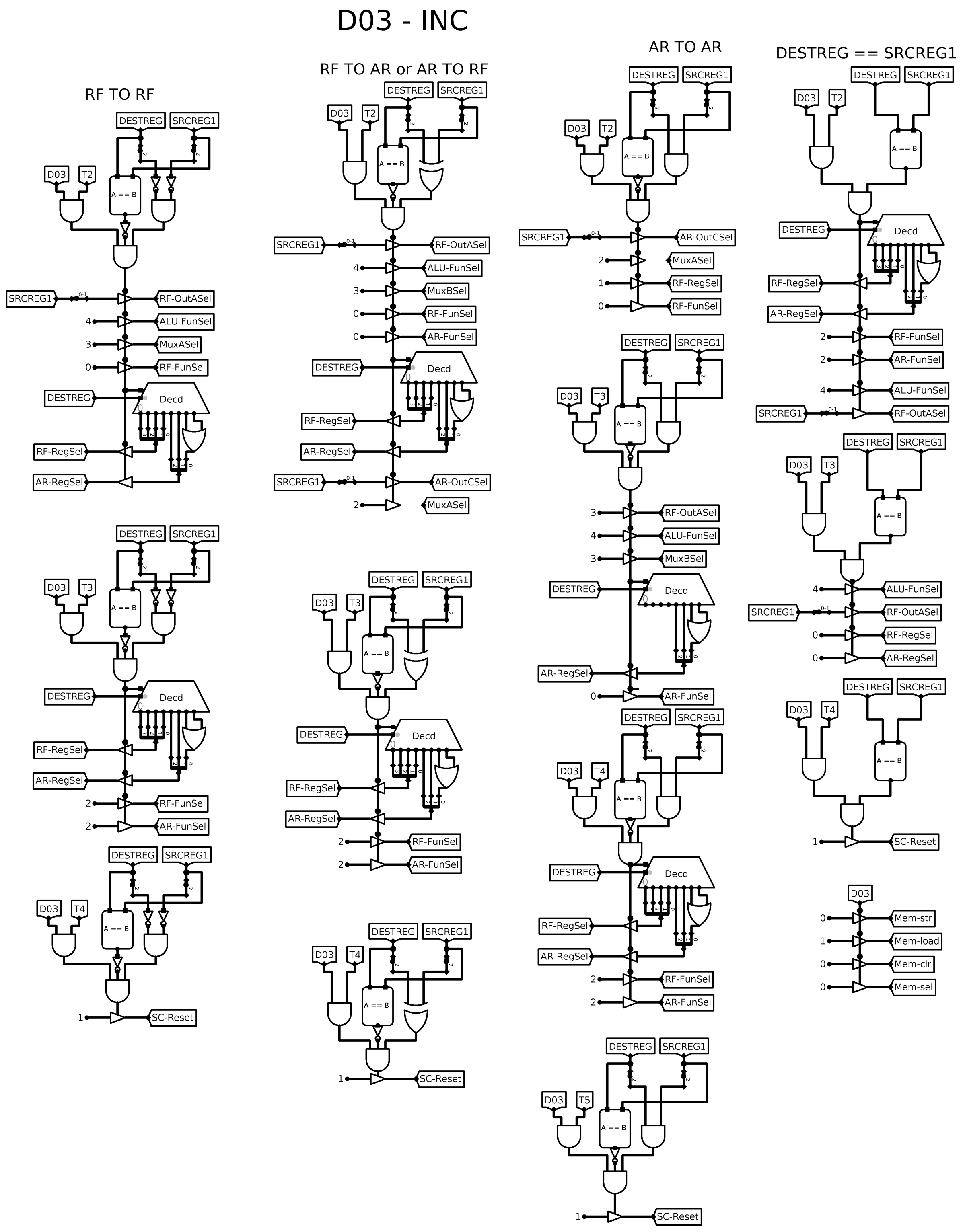


Figure 7: Increment Operation (INC) Control Circuit

2.4.5 BRA OPCODE: 0x04

The fifth operation and two subsequent operations are similarly the same. Because what they do is branching. The addressing mode is 1. For this operation since the instruction is an address reference it does not need to decoded. The ALU FunSel is made to apply the BRA operation to the given value.

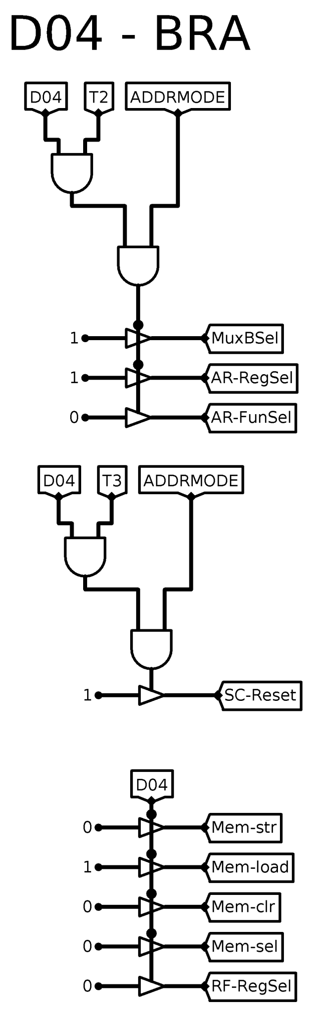


Figure 8: BRA Control Circuit

2.4.6 BEQ OPCODE: 0x05

The sixth operation is BEQ. If zero flag equals to 1, then it assigns the value to PC. Its addressing mode is also 1. The instruction does not need to decoded because it is an address reference. The ALU FunSel is made to apply the BEQ operation to the given value.

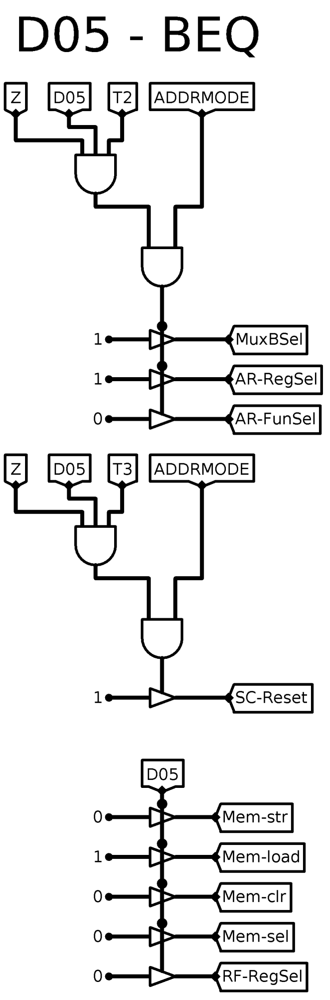


Figure 9: BEQ Control Circuit

2.4.7 BNE OPCODE: 0x06

The seventh operation is BNE. If zero flag equals to 0, then it assigns the value to PC. Its addressing mode is also 1. The instruction does not need to decoded because it is an address reference. The ALU FunSel is made to apply the BEQ operation to the given value.

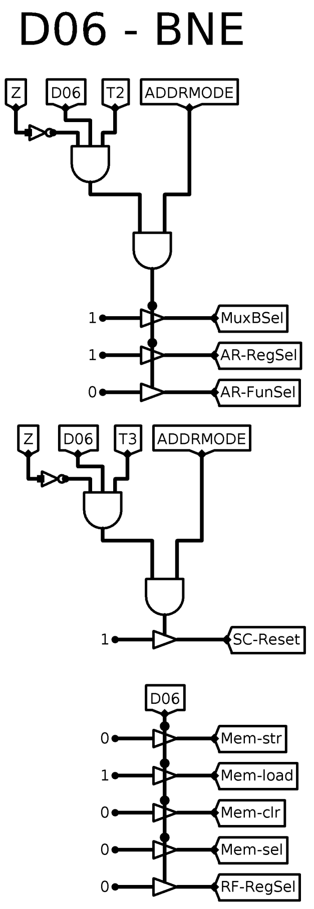


Figure 10: BNE Control Circuit

2.4.8 LSL OPCODE: 0x07

The eighth operation is a logical left shift operation. For the logical shift operation, the instruction is decoded as shown in Table 2. This operation simply shifts the value to the left by one bit and changes the least significant bit to a zero. The structure behaves differently for 4 possible case such as SRCREG1 be from AR or RF and DESTREG be from AR or RF.

To explain the structure of control circuit used for LSL and LSR, the operations that are done in particular timing cycles should be stated.

First of all, the type of the operation is selected by checking the most significant bit of the SRCREG1 and DESTREG and the operation applies one of the 4 case that is explained below.

- RF to RF:
  - T2:
    - The SRCREG1's first two bit is assigned to RF-OutASel.

        * The LSL operation is selected from the ALU by assigning the 1010 value to the ALU-FunSel.
        * To write the result that is coming from the ALU to the RF the MuxASel is assigned to 11.
        * To select the load operation of the RF the RF-FunSel is assigned to 00.
        * By a certain subcircuit that is designed to select the right register the RF-RegSel is assigned to the right value.
    - T3:
        * The sequence counter is reset.

- RF to AR:
    - T2:
        * The SRCREG1's first two bit is assigned to TF-OutASel.
        * The LSL operation is selected from the ALU by assigning the 1010 value to the ALU-FunSel.
        * To write the result that is coming from the ALU to the AR the MuxBSel is assigned to 11.
        * To select the load operation of the AR the AR-FunSel is assigned to 00.
        * By the subcircuit that is mentioned in the RF to RF section, the right register of the AR-RegSel is assigned to the right value.
        * The register that has been loaded is selected to be used in future operations by setting AR-OutCSel to SRCREG1's first two bit.
        * The MuxASel is assigned to 10 to select the value that is coming from the previously loaded AR.
    - T3:
        * The sequence counter is reset.

- AR to RF:
    - T2:
        * The AR-OutCSel is set to SRCREG1's first two bit.
        * The MuxASel is set to 10 to select the value that is coming from the AR.
        * The RF-FunSel is set to 00 to load the given value to the selected register file.
        * By the subcircuit, the right register of the RF is selected by checking the DESTREG.

- T3:
  - The RF-OutASel is set to DESTREG's first two bit to select the register that is previously loaded.
  - The LSL operation is selected from the ALU by assigning ALU-FunSel to 1010.
  - The MuxASel is set to 11 to select the value that is coming from the ALU.
  - The RF-FunSel is set to 00 to apply the load operation.
  - By the subcircuit, the right register of the RF is selected by checking the DESTREG.
- T4:
  - The sequence counter is reset.

- AR to AR:
  - T2:
    - The AR-OutCSel is set to SRCREG1's first two bit.
    - The MuxASel is set to 10 to select the value that is coming from the AR.
    - The RF-RegSel is set to 0001 to load the temporary value to the register.(The R0 register has been selected to use as a temporary register.)
    - The RF-FunSel is set to 00 to apply the load operation.
  - T3:
    - The RF-OutASel is set to 11 to select the R0(temporary register).
    - The ALU-FunSel is set to 1010 to apply the LSL operation to the value.
    - The MuxBSel is set to 11 select the value that is coming from the ALU.
    - By the subcircuit the right register of the AR is selected by checking the DESTREG and assigning the right value to the AR-RegSel.
    - The AR-FunSel is set to 00 to apply the load operation.
  - T4:
    - The sequence counter is reset.

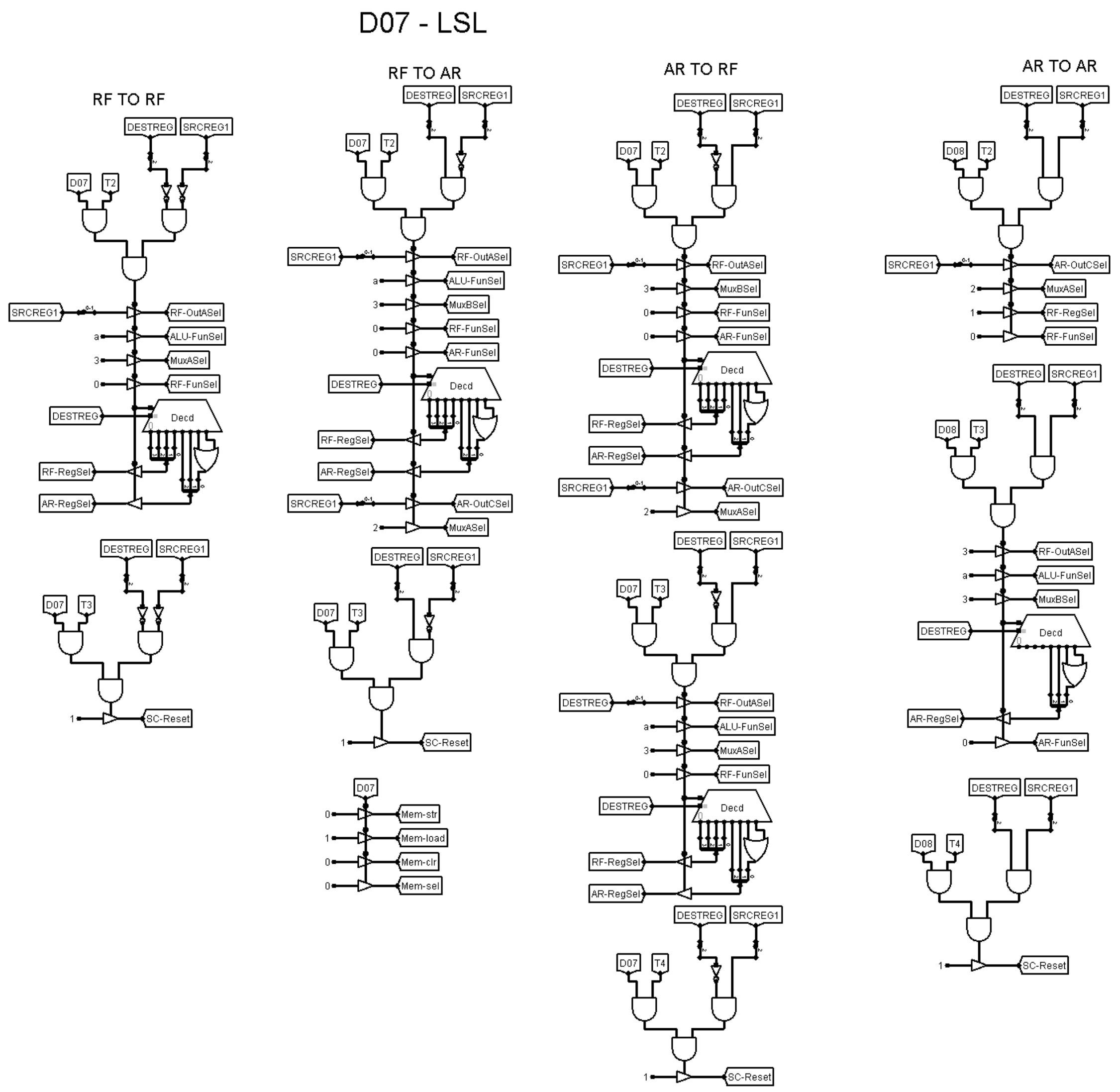


Figure 11: Logical Shift Left (LSL) Control Circuit

2.4.9 LSR OPCODE: 0x08

The ninth operation is the LSR operation. Since this operation is very similar to the LSL(D07) operation, the same structure with very small differences is used. The only difference is to make the LSR operation to the particular value in the ALU. The ALU-FunSel signal is simply changed from 1010 to 1011. With that the LSR operation is successfully implemented.

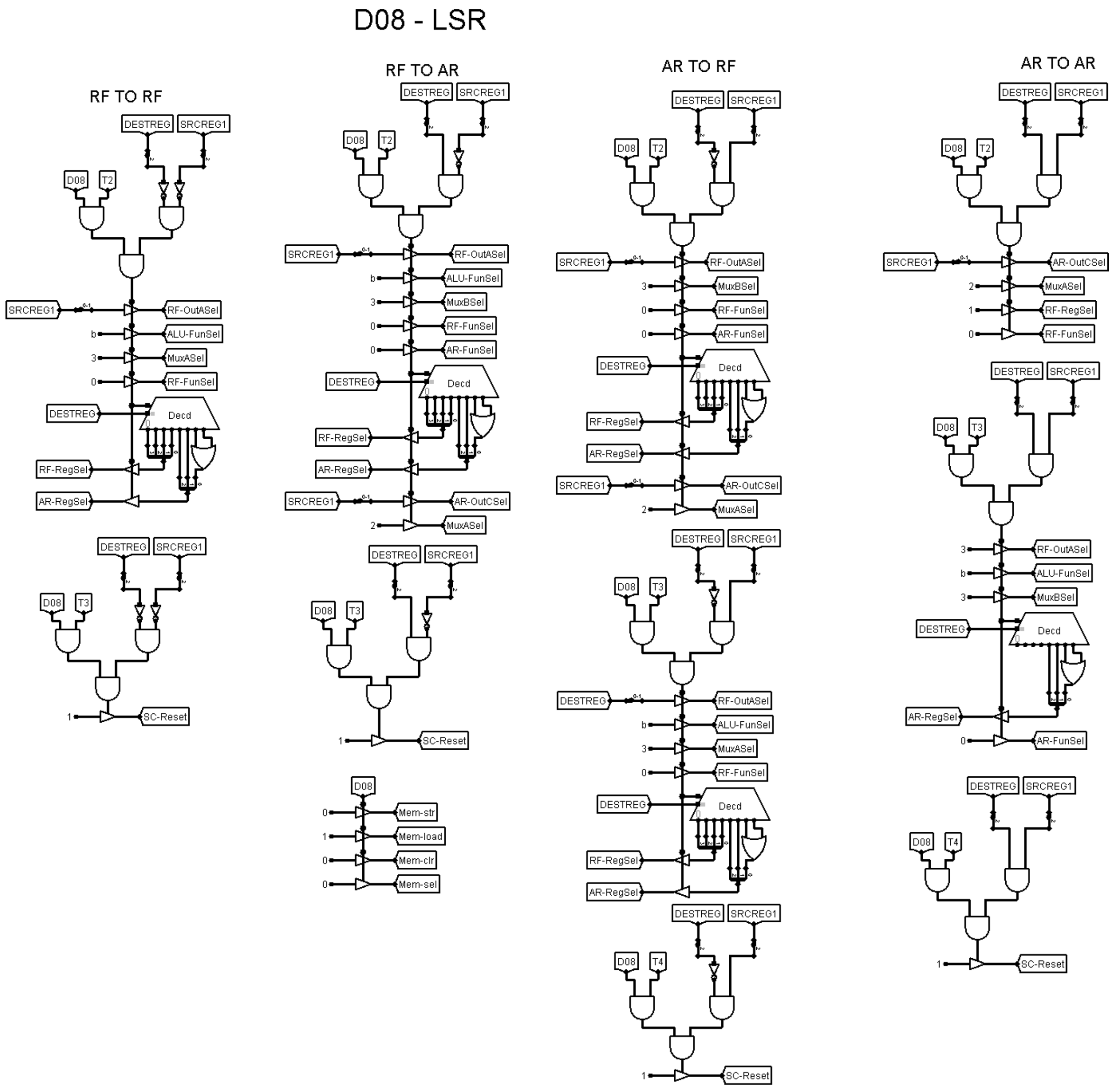


Figure 12: Logical Shift Right (LSR) Control Circuit

2.4.10 LD OPCODE: 0x09

The tenth operation is LOAD(LD). This operation enables that loading something from memory into a register.

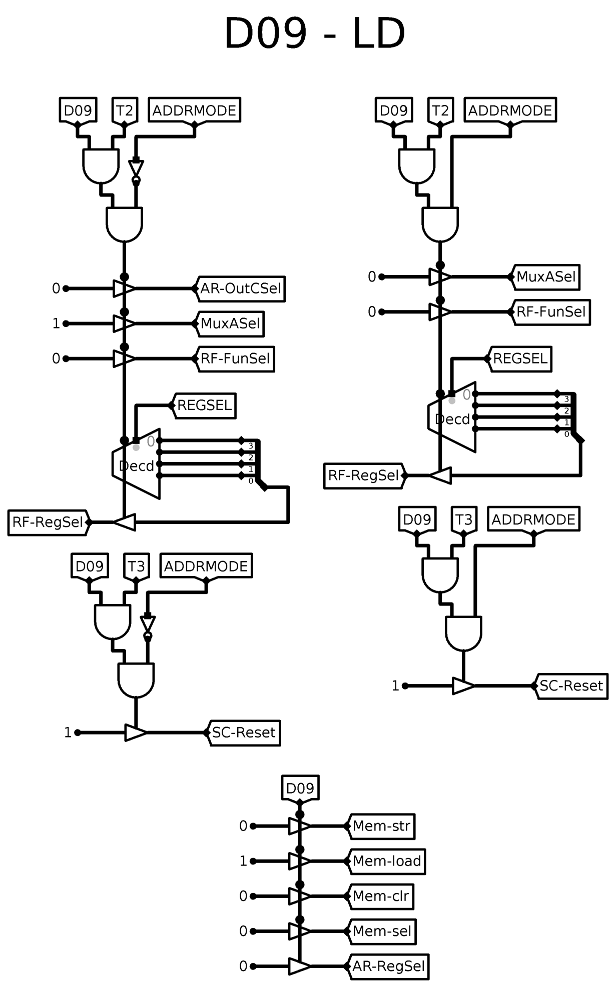


Figure 13: Load(LD) Control Circuit

2.4.11 ST OPCODE: 0x0A

The eleventh operation is STORE(ST). This operation enables that storing something from a register to a memory address.

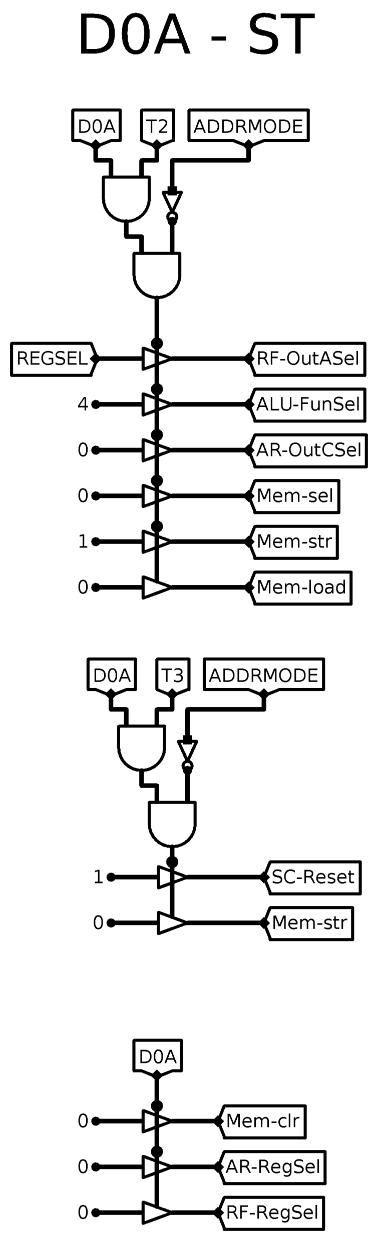


Figure 14: Store(ST) Control Circuit

2.4.12 MOV OPCODE: 0x0B

Moving data along the organization is rather simple, output values are taking from the $SRCREG(0-1)$ which is given to both $OutASel$ of the register file and $OutSel$ of the address register file. Data is channeled through multiplexers by giving right control signals.

Moving data from adress register file to address register file takes one more cycle, data has to be moved on register file first then should come back to the address register file since there is no direct connection between output and input of the address register file.

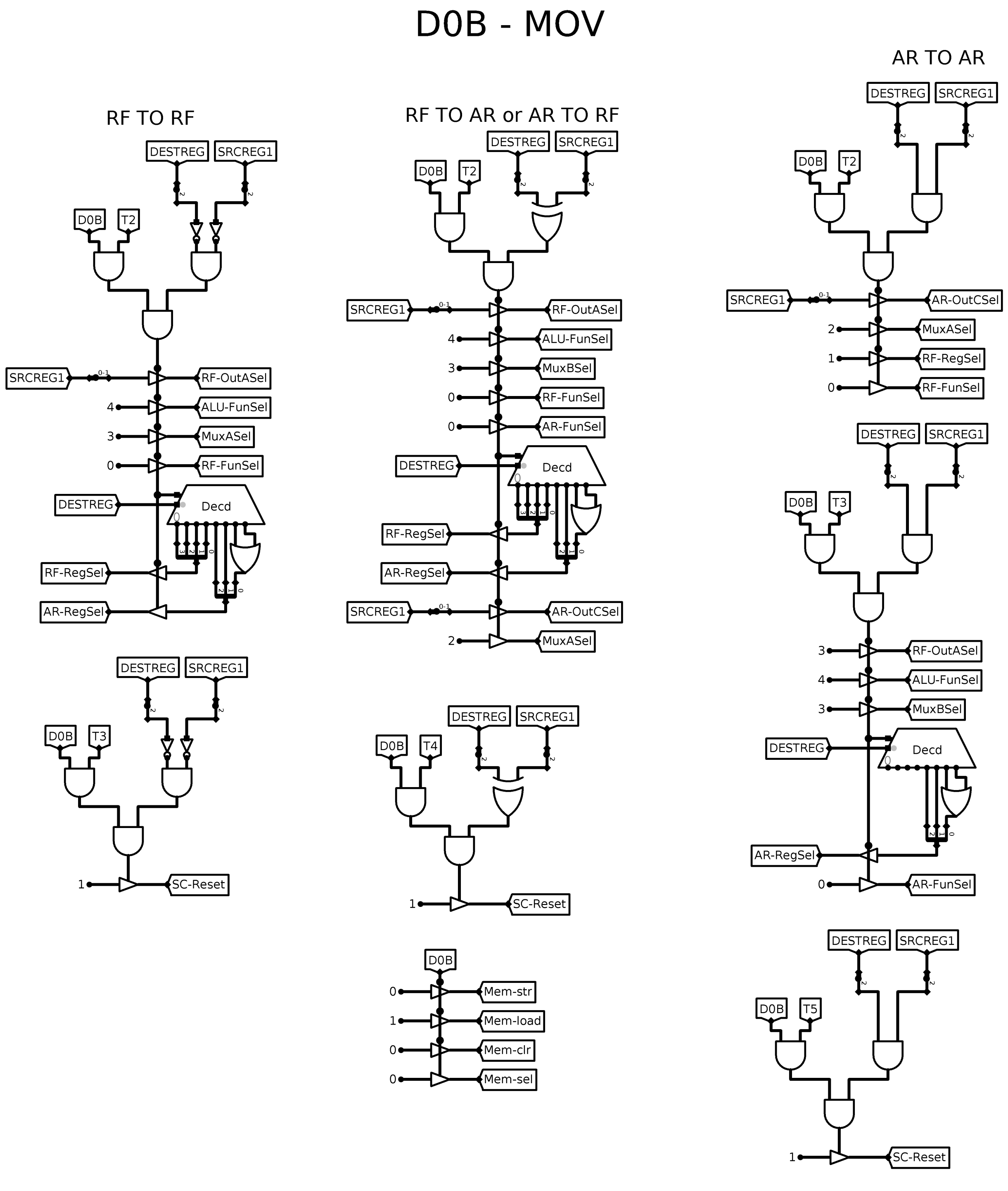


Figure 15: MOV operation (MOV) Control Circuit

2.4.13 NOT OPCODE: 0x0C

The twelfth operation is the NOT operation. Since this operation is very similar to the LSL(D07) and LSR(D08) operations, the same structure with very small differences is used. The only difference is to make the ALU FunSel to apply the NOT operation to the given value. The ALU-FunSel signal is simply changed to 0010. With that the NOT operation is successfully implemented.

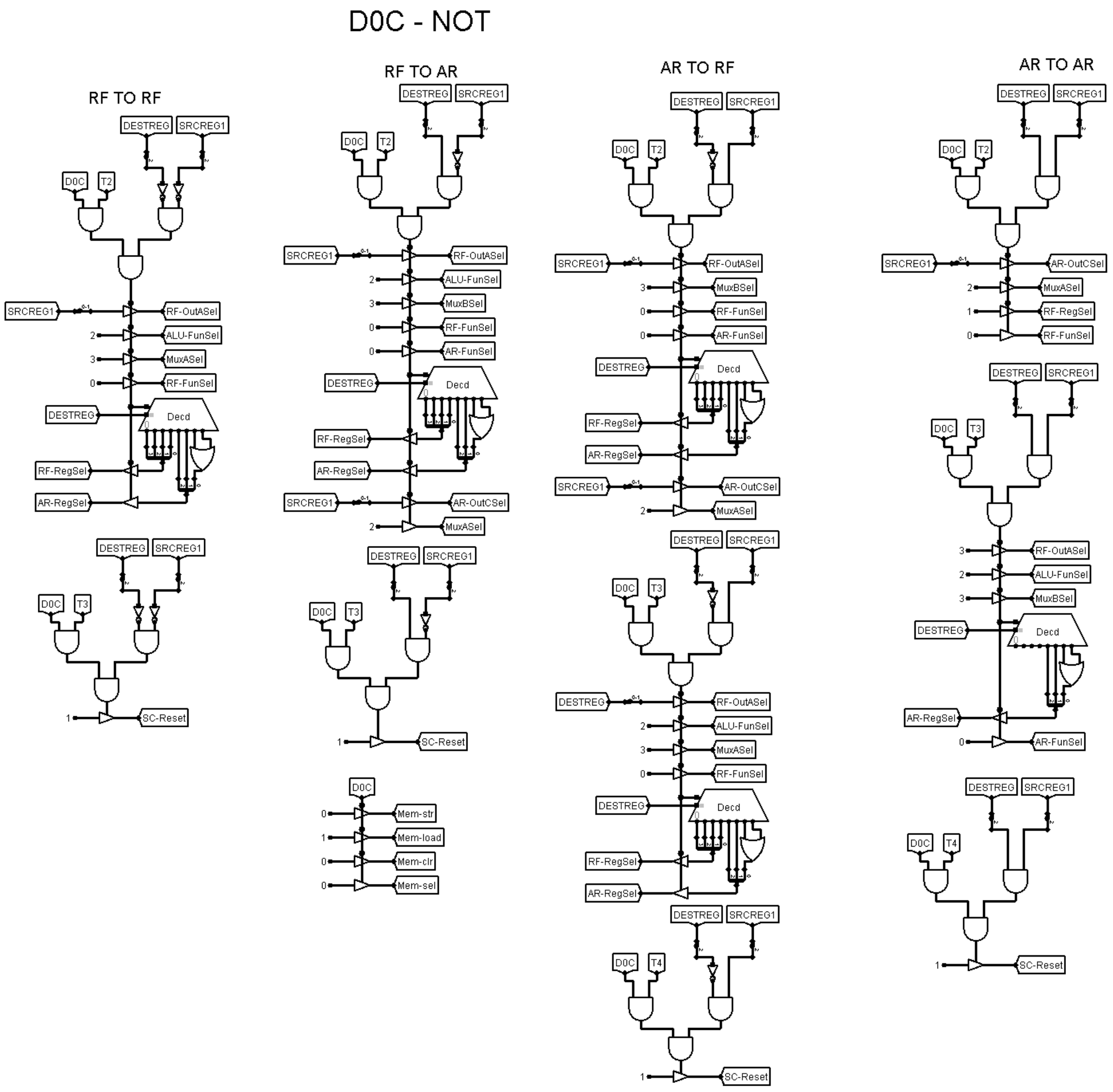


Figure 16: Not Operation (NOT) Control Circuit

2.4.14 OR OPCODE: 0x0D

This operation's control signals are exactly the same with the addition operation except for the ALU-FunSel, which is given 0001 in order to obtain OR operation.

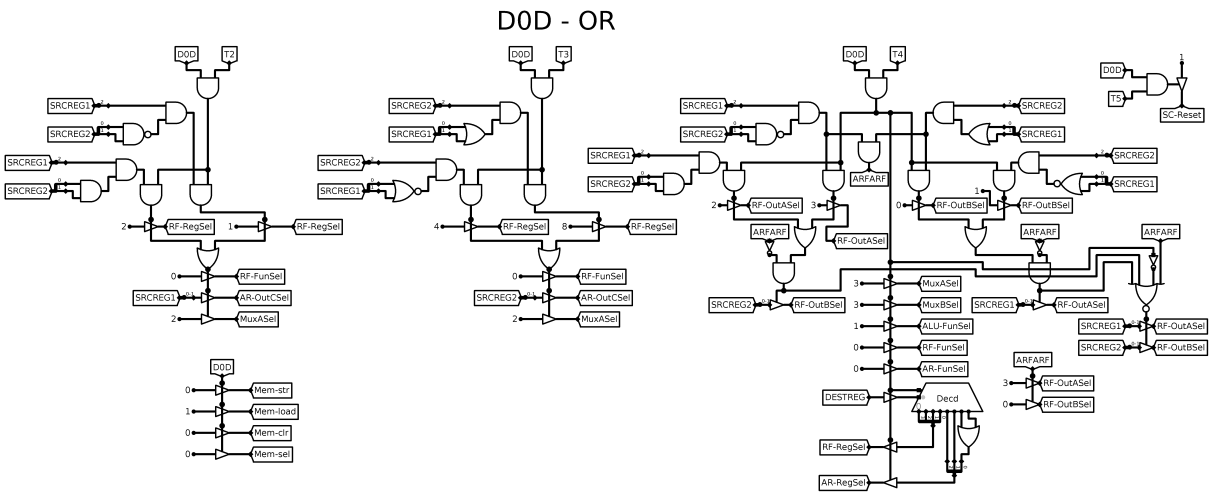


Figure 17: OR Operation (OR) Control Circuit

2.4.15 AND OPCODE: 0x0E

This operation's control signals are completely same with the addition operation only the ALU-FunSel differs, which is given 0000 in order to obtain AND operation.

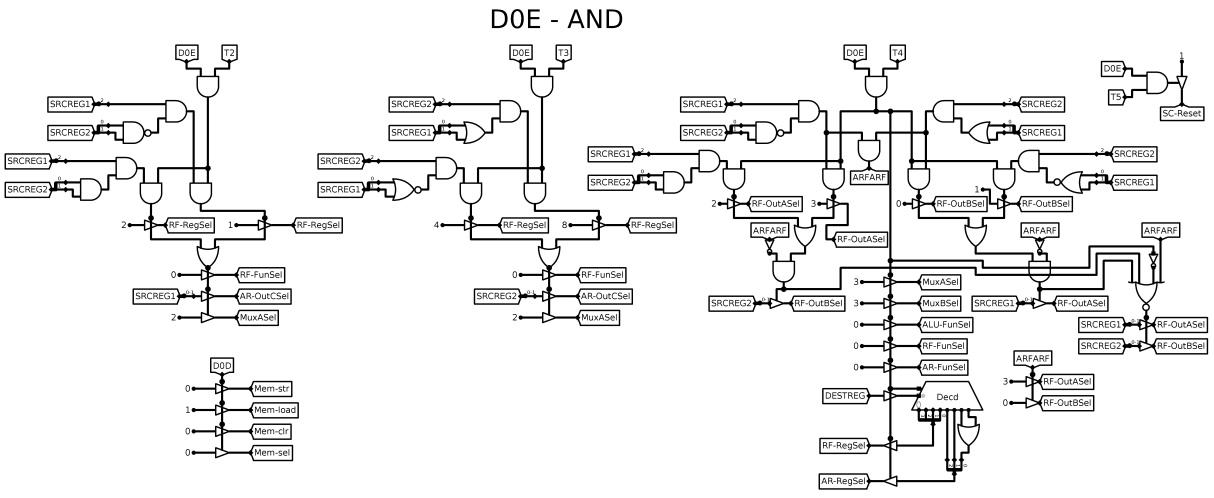


Figure 18: AND Operation (AND) Control Circuit

3 DISCUSSION

Control unit that handles all processor control signals is the brain of the CPU. CPU is the brain of computer so control unit is also the brain of the computer. Each program in the computer has many instructions and these set of instructions are continuously fetched, decoded and executed via the control unit. The architecture in this project can perform 15 different operations. Each program that can be executed with these instructions through this architecture.

4 CONCLUSION

By doing this project, experience on hard-wired implementing of basic computer and processing operations are gained.

REFERENCES

[1] Overleaf documentation https://tr.overleaf.com/learn.

[2] Detailed info on writing reports https://projects.ncsu.edu/labwrite/res/res-studntintro-labparts.html.

[3] Website lets collabration over projects. https://github.com/.