BLG222E Computer Organization

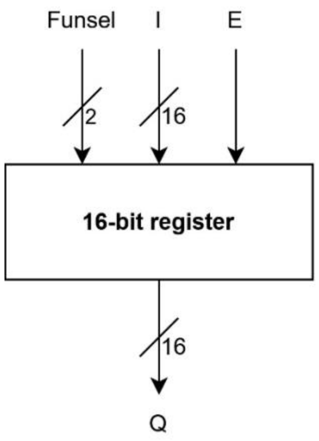
Project 1

Due Date: 27.07.2025 23:59

In this project, registers and register files will be designed and implemented. It has 4 parts. Design each part **as a module**, so that it can be reused in other parts. **You must simulate each module for each combination of input.**

(Part-1) Design 16-bit and 32-bit registers that works as follows.

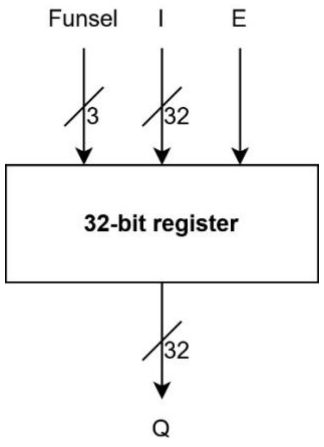
(Part-1a) Design a **16-bit register**. This register has 4 functionalities that are controlled by 2-bit control signals **(FunSel)** and an enable input **(E).** The graphic symbol of the registers and the characteristic table are shown in Figure 1. Symbol **ϕ** means **"don't care"**. This register will be used in later parts.



E	FunSel	Q^+
0	ϕ	Q (Retain value)
1	00	Q-1 (Decrement)
1	01	Q+1 (Increment)
1	10	I (Load)
1	11	0 (Clear)

Figure 1: Graphic symbol of 16-bit register and the characteristic table, respectively.

(Part-1b) Design a **32-bit register** that has 8 functionalities that are controlled by 3-bit control signals **(FunSel)** and an enable input **(E).** This register also will be used in later parts.

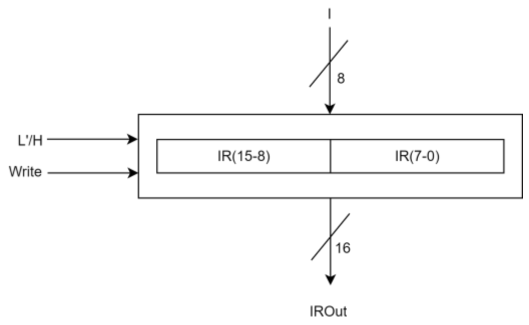


E	FunSel	Q^+
0	ϕ	Q (Retain value)
1	000	Q-1 (Decrement)
1	001	Q+1 (Increment)
1	010	I (Load)
1	011	0 (Clear)
1	100	Q (31-8) ← Clear, Q (7-0) ← I (7-0)
1	101	Q (31-16) ← Clear, Q (15-0) ← I (15-0)
1	110	Q (31-8) ← Q (23-0) (8-bit Left Shift) Q (7-0) ← I (7-0)
1	111	Q (31-16) ← Sign Extend (I (15)) (16-bit sign extend) Q (15-0) ← I (15-0)

Figure 2: Graphic symbol of 32-bit register and the characteristic table, respectively.

(Part-2) Design a register file (a structure that contains many registers) that works as follows.

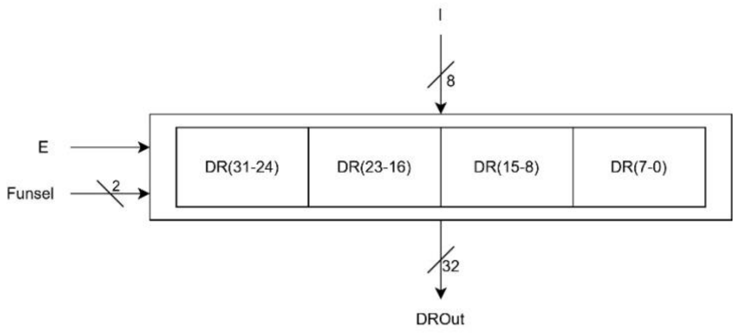
(Part-2a) Design a 16-bit **Instruction Register (IR)** register whose block diagram and function table are given in Figure 3. This register can store 16-bit binary data. However, the input of this register file is only 8 bits. Therefore, using the 8-bit input bus, you can load either the lower (bits 7-0) or higher (bits 15-8) half. This decision is made by the **L'H** signal.



L'H	**Write**	**IR⁺**
ϕ	0	IR (Retain value)
0	1	IR (7-0) ← I (Load LSB)
1	1	IR (15-8) ← I (Load MSB)

Figure 3: Graphic symbol of IR register and its characteristic table, respectively.

(Part-2b) Design a 32-bit **Data Register (DR)** register whose block diagram and function table are given in Figure 4. This register can store 32-bit binary data. The input data bus of the DR register file, similar to the IR register, is limited to 8 bits. Therefore, only 8 bits of data can be loaded into the register per clock cycle, requiring four cycles to fully load 32 bits of data.



E	**FunSel**	**DR⁺**
0	ϕ	DR (Retain value)
1	00	DR (31-8) ← Sign Extend (I (7)) (8-bit sign extend) DR (7-0) ← I (7-0)
1	01	DR (31-8) ← Clear, DR (7-0) ← I (7-0)
1	10	DR (31-8) ← DR (23-0) (8-bit Left Shift) DR (7-0) ← I (7-0)
1	11	DR (23-0) ← DR (31-8) (8-bit Right Shift) DR (31-24) ← I (7-0)

Figure 4: Graphic symbol of DR register and its characteristic table, respectively.

(Part-2c) Design the **Register File (RF)** shown in Figure 5 which consists of four 32-bit general purpose registers: **R1**, **R2**, **R3**, **R4**, and four 32-bit scratch registers: **S1**, **S2**, **S3**, and **S4.** The details of inputs and outputs are as follows.

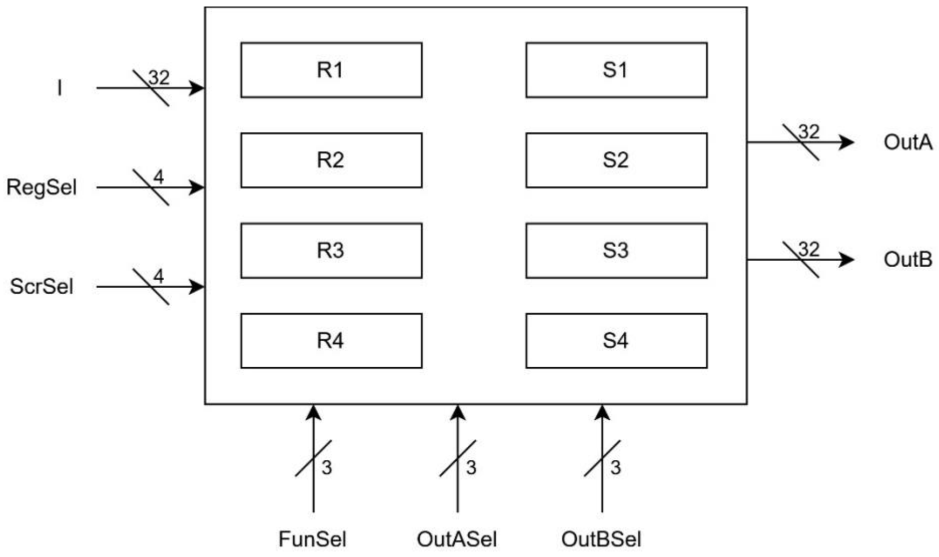


Figure 5: 32-bit general purpose and scratch registers, inputs, and outputs.

OutASel and **OutBSel** are used to feed to output lines **OutA** and **OutB**, respectively. 32 bits of selected registers are output to **OutA** and **OutB**. Table 1 shows the selection of output registers based on the **OutASel** and **OutBSel** control inputs.

Table 1: OutASel and OutBSel controls.

OutASel	OutA
000	R1
001	R2
010	R3
011	R4
100	S1
101	S2
110	S3
111	S4

OutBSel	OutB
000	R1
001	R2
010	R3
011	R4
100	S1
101	S2
110	S3
111	S4

RegSel and **ScrSel** are 4-bit signals that select the registers to apply the function that is determined by the 3-bit **FunSel** signal which is given in Table 2. The selected registers by **RegSel** and **ScrSel** are shown in Tables 3 and 4, respectively.

Table 2: FunSel Control Input.

FunSel	R_x^+ (Next State)
000	R_x-1 (Decrement)
001	R_x +1 (Increment)
010	I (Load)
011	0 (Clear)
100	R_x (31-8) $\leftarrow$ Clear, R_x (7-0) $\leftarrow$ I (7-0)
101	R_x (31-16) $\leftarrow$ Clear, R_x (15-0) $\leftarrow$ I (15-0)

110	R_x (31-8) ← R_x (23-0) (8-bit Left Shift) R_x (7-0) ← I (7-0)
111	R_x (31-16) ← Sign Extend (I (15)) (16-bit sign extend) R_x (15-0) ← I (15-0)

Table 3: RegSel Control Input

RegSel	Enable General Purpose Registers	RegSel	Enable General Purpose Registers
0000	NO general purpose register is enabled. (All registers retain their values.)	1000	Only R1 is enabled. (Function selected by FunSel will be applied to R1.)
0001	Only R4 is enabled. (Function selected by FunSel will be applied to R4.)	1001	R1 and R4 are enabled. (Function selected by FunSel will be applied to R1 and R4.)
0010	Only R3 is enabled. (Function selected by FunSel will be applied to R3.)	1010	R1 and R3 are enabled. (Function selected by FunSel will be applied to R1 and R3.)
0011	R3 and R4 are enabled. (Function selected by FunSel will be applied to R3 and R4.)	1011	R1, R3, and R4 are enabled. (Function selected by FunSel will be applied to R1, R3, and R4.)
0100	Only R2 is enabled. (Function selected by FunSel will be applied to R2.)	1100	R1 and R2 are enabled. (Function selected by FunSel will be applied to R1 and R2.)
0101	R2 and R4 are enabled. (Function selected by FunSel will be applied to R2 and R4.)	1101	R1, R2, and R4 are enabled. (Function selected by FunSel will be applied to R1, R2, and R4.)
0110	R2 and R3 are enabled. (Function selected by FunSel will be applied to R2 and R3.)	1110	R1, R2 and R3 are enabled. (Function selected by FunSel will be applied to R1, R2, and R3.)
0111	R2, R3, and R4 are enabled. (Function selected by FunSel will be applied to R2, R3, and R4.)	1111	All general purpose registers are enabled. (Function selected by FunSel will be applied to R1, R2, R3 and R4.)

Table 4: ScrSel Control Input

ScrSel	Enable General Purpose Registers	ScrSel	Enable General Purpose Registers
0000	NO general purpose register is enabled. (All registers retain their values.)	1000	Only S1 is enabled. (Function selected by FunSel will be applied to S1.)
0001	Only S4 is enabled. (Function selected by FunSel will be applied to S4.)	1001	S1 and S4 are enabled. (Function selected by FunSel will be applied to S1 and S4.)
0010	Only S3 is enabled. (Function selected by FunSel will be applied to S3.)	1010	S1 and S3 are enabled. (Function selected by FunSel will be applied to S1 and S3.)
0011	S3 and S4 are enabled. (Function selected by FunSel will be applied to S3 and S4.)	1011	S1, S3, and S4 are enabled. (Function selected by FunSel will be applied to S1, S3, and S4.)
0100	Only S2 is enabled. (Function selected by FunSel will be applied to S2.)	1100	S1 and S2 are enabled. (Function selected by FunSel will be applied to S1 and S2.)
0101	S2 and S4 are enabled. (Function selected by FunSel will be applied to S2 and S4.)	1101	S1, S2, and S4 are enabled. (Function selected by FunSel will be applied to S1, S2, and S4.)
0110	S2 and S3 are enabled. (Function selected by FunSel will be applied to S2 and S3.)	1110	S1, S2, and S3 are enabled. (Function selected by FunSel will be applied to S1, S2, and S3.)
0111	S2, S3, and S4 are enabled. (Function selected by FunSel will be applied to S2, S3, and S4.)	1111	All general purpose registers are enabled. (Function selected by FunSel will be applied to S1, S2, S3, and S4.)

(Part-2d) Design the **Address Register File (ARF)** system shown in Figure 4 which consists of three 16-bit address registers: **program counter (PC)**, **address register (AR)**, **and stack pointer (SP)**.

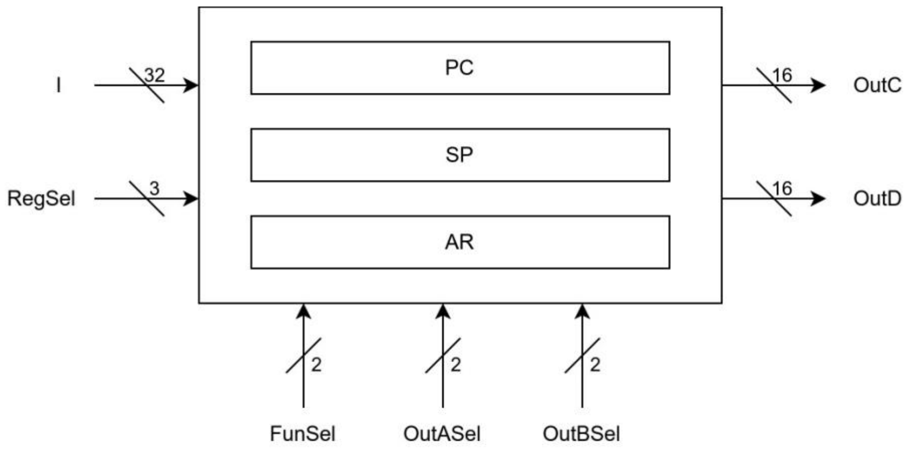


Figure 6: 16-bit address registers, inputs, and outputs.

OutCSel and **OutDSel** are used to feed output lines **OutC** and **OutD**, respectively. 16 bits of the selected registers are output to **OutC** and **OutD**. Table 5 shows the selection of output registers based on the **OutCSel** and **OutDSel** control inputs.

Table 5: OutCSel and OutDSel controls.

OutCSel	OutC
00	PC
01	SP
10	AR
11	AR

OutDSel	OutD
00	PC
01	SP
10	AR
11	AR

RegSel is 3-bit signal that select the registers to apply the function that is determined by the 2-bit **FunSel** signal which is given in Table 6. The selected registers by **RegSel** are shown in Table 7.

Table 6: FunSel Control Input.

FunSel	R_x^+ (Next State)
00	R_x-1 (Decrement)
01	R_x +1 (Increment)
10	I (Load)
11	0 (Clear)

Table 7: RegSel Control Input

ScrSel	Enable General Purpose Registers	ScrSel	Enable General Purpose Registers
000	NO registers is enabled. (All registers retain their values.)	100	Only PC is enabled. (Function selected by FunSel will be applied to PC.)
001	Only AR is enabled. (Function selected by FunSel will be applied to AR.)	101	PC and AR are enabled. (Function selected by FunSel will be applied to PC.)
010	Only SP is enabled. (Function selected by FunSel will be applied to SP.)	110	PC and SP are enabled. (Function selected by FunSel will be applied to PC and SP.)
011	SP and AR are enabled. (Function selected by FunSel will be applied to SP and AR.)	111	All registers are enabled. (Function selected by FunSel will be applied to PC, SP, and AR.)

(Part-3) Design an Arithmetic Logic Unit (ALU) that has two 32-bit inputs, a 32-bit output, and a 4-bit output for **z**ero, **n**egative, **c**arry, and **o**verflow flags. The ALU is shown in Figure 7. The ALU functions and the flags that will be updated (i.e., **-** means that the flag will not be affected and **+** means that the flag changes based on the ALUOut) are given in Table 8 when **WF** (Write Flag) is 1.

- **FunSel** selects the function of the ALU.
- **ALUOut** shows the result of the operation that is selected by **FunSel** and applied on A and/or B inputs.
- **Arithmetic operations** are done using **2's complement** logic.
- **Z (zero)** bit is set if **ALUOut** is zero (e.g. when **NOT B** is zero).
- **C (carry)** bit is set if **ALUOut** sets the carry (e.g. when **LSL A** produces carry)
- **N (negative)** bit is set if the ALU operation generates a negative result (e.g. when **A-B** results in a negative number).
- **O (overflow)** bit is set if an overflow occurs (e.g. when **A+B** results in an overflow).
- Note that **Z|C|N|O** flags are stored in a **register**! The **Z** flag is the **MSB** of the register, and the **O** flag is the **LSB** of the register.

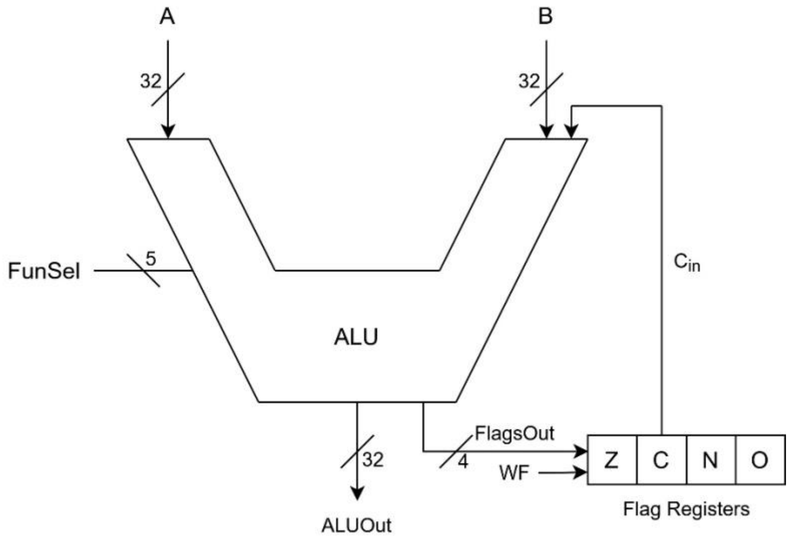


Figure 7: Arithmetic Logic Unit graphic.

Table 8: Characteristic table of ALU.

FunSel	ALUOut	Z	C	N	O
00000	A (16-bit)	+	-	+	-
00001	B (16-bit)	+	-	+	-
00010	NOT A (16-bit)	+	-	+	-
00011	NOT B (16-bit)	+	-	+	-
00100	A + B (16-bit)	+	+	+	+
00101	A + B + Carry (16-bit)	+	+	+	+
00110	A – B (16-bit)	+	+	+	+
00111	A AND B (16-bit)	+	-	+	-
01000	A OR B (16-bit)	+	-	+	-
01001	A XOR B (16-bit)	+	-	+	-
01010	A NAND B (16-bit)	+	-	+	-
01011	LSL A (16-bit)	+	+	+	-
01100	LSR A (16-bit)	+	+	+	-
01101	ASR A (16-bit)	+	-	-	-
01110	CSL A (16-bit)	+	+	+	-
01111	CSR A (16-bit)	+	+	+	-
10000	A (32-bit)	+	-	+	-
10001	B (32-bit)	+	-	+	-
10010	NOT A (32-bit)	+	-	+	-
10011	NOT B (32-bit)	+	-	+	-
10100	A + B (32-bit)	+	+	+	+
10101	A + B + Carry (32-bit)	+	+	+	+
10110	A – B (32-bit)	+	+	+	+
10111	A AND B (32-bit)	+	-	+	-
11000	A OR B (32-bit)	+	-	+	-
11001	A XOR B (32-bit)	+	-	+	-
11010	A NAND B (32-bit)	+	-	+	-
11011	LSL A (32-bit)	+	+	+	-
11100	LSR A (32-bit)	+	+	+	-
11101	ASR A (32-bit)	+	-	-	-
11110	CSL A (32-bit)	+	+	+	-
11111	CSR A (32-bit)	+	+	+	-

(**C**ircular | **A**rithmetic | **L**ogical) Shift (**L**eft | **R**ight) operations are depicted in Figures 8, 9, and 10.

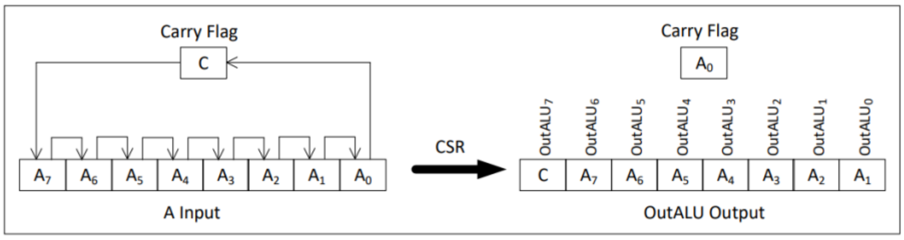


Figure 8: Circular shift operation.

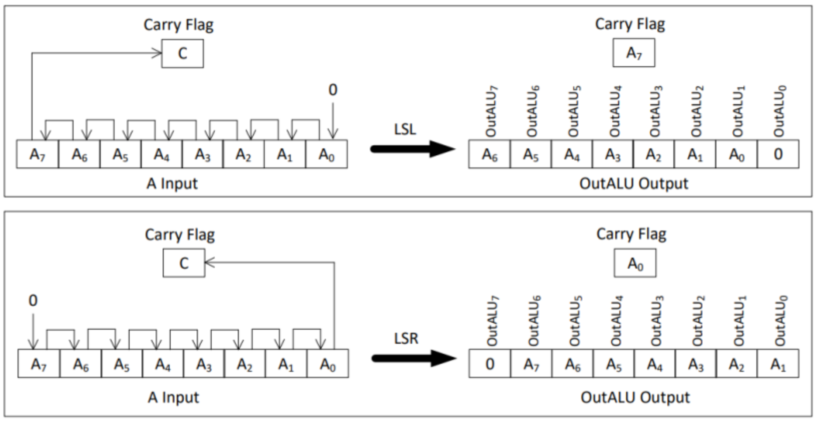


Figure 9: Logical shift operation.

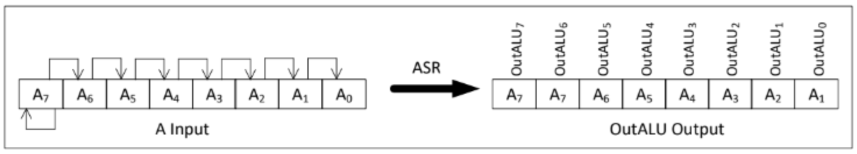


Figure 10: Arithmetic shift

(Part-4) Implement the organization in Figure 11. Please note that **the whole system uses the same single clock.**

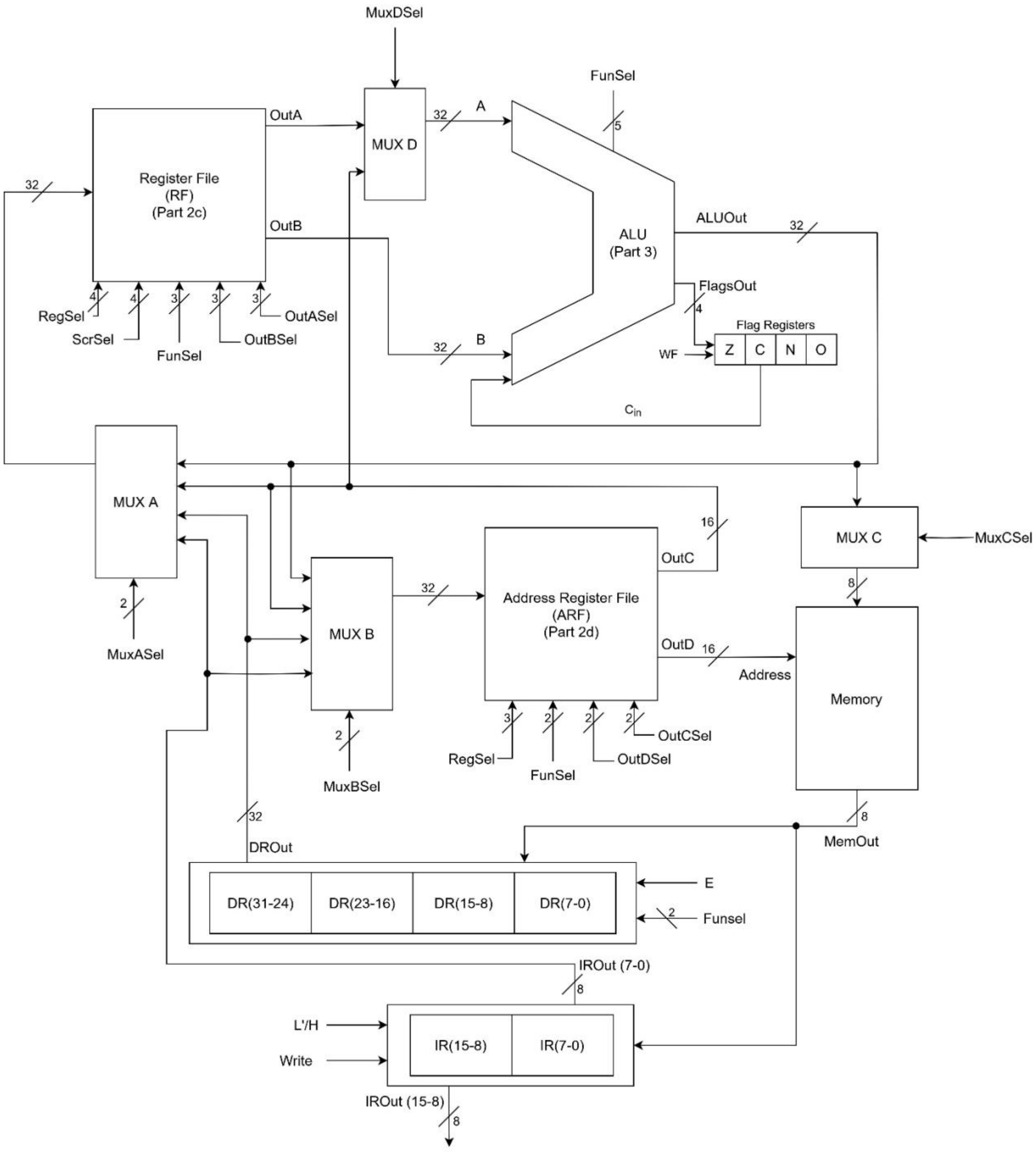


Figure 11: ALU System.

Table 9: Multiplexers of ALU Systems.

MuxASel	**MuxAOut**
00	ALUOut
01	ARF OutC
10	DROut
11	IROut (7:0)

MuxBSel	**MuxAOut**
00	ALUOut
01	ARF OutC
10	DROut
11	IROut (7:0)

MuxCSel	**MUXCOut**
00	ALUOut (7-0)
01	ALUOut (15-8)
10	ALUOut (23-16)
11	ALUOut (31-24)

MuxDSel	**MuxDOut**
0	RF OutA
1	ARF OutC

Submission:

Implement your design in Verilog HDL, and upload a single compressed (zip) file to Ninova before the deadline. Only one student from each group should submit the project file (select one member of the group as the group representative for this purpose and note his/her student ID). Example submission file and module name declarations are given as attachments. This compressed file should contain your modules file (.v) for each part, the given simulation file (.v), and a report that contains:

- the number&names of the students in the group
- list of control inputs and corresponding functions for your design
- explanations for each constructed part
- task distribution of each group member

Group work is expected for this project. All members of the group must design together. You must ensure that all modules work properly. After designing your project, you can check your results by running the given simulation files. You are expected to get results by running the given .bat file. **The project will be evaluated only using simulation files by running Run.bat file on Vivado 2017.4. There will not be any partial grading for the designs. If your codes get any error, you will not get any grades for that part so make sure that your codes are working with the given simulation files. For detailed information about testing and submission procedures, check the provided *Running Simulation Tests.pdf* file. There will not be any demonstration sessions.** You can ask your questions through the **Message Board,** so do not ask questions through email.