BLG222E Computer Organization

Project 1

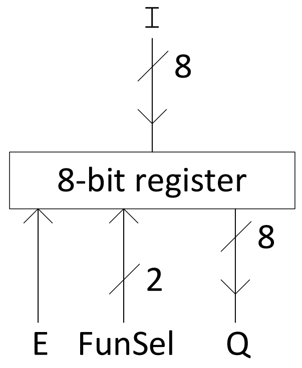
Due Date: 19.05.2021 23:59

In this project, registers and register files will be designed and implemented. It has 4 parts. Design each part **as a library unit**, so that it can be reused in other parts. **You can use any available combinational/sequential logisim units in your projects.**

(Part-1) Design 2 different types of registers: **(1)** 8-bit register and **(2)** 16-bit register.

(Part-1a) The 8-bit register has 4 functionalities that are controlled by 2-bit control signals (**FunSel**) and an enable input (**E**).

The block diagram of the 8-bit register and the function table is shown in Figure 1. Symbol ϕ means don't care. Build the register as a library in logisim software so that you can use them in other parts.

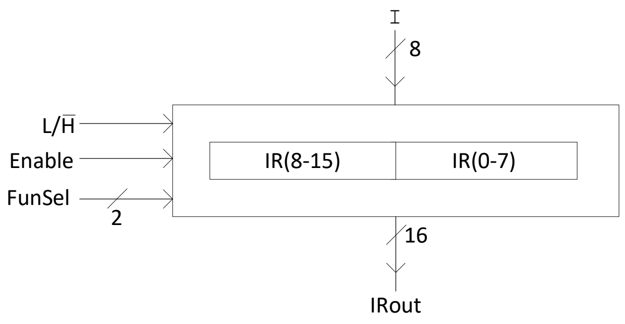


E	FunSel	Q^+	
0	ϕ	Q	(Retain Value)
1	00	Q-1	(Decrement)
1	01	Q+1	(Increment)
1	10	I	(Load)
1	11	0	(Clear)

Figure 1: Block Diagram of the registers (Left) and the function table (Right)

(Part-1b) Design an 16-bit **IR** register whose block diagram and function table are given in Figure 2.

This register can store 16 bit binary data. However, the input of this register file is only 8 bits. Hence, using the 8 bit input you can load either the lower (bits 7-0) or higher (bits 15-8) half. This is determined by $\boldsymbol{L/\overline{H}}$ signal.



$L/\overline{H}$	Enable	FunSel	IR^+	
ϕ	0	ϕϕ	IR	(Retain Value)
ϕ	1	00	IR - 1	(Decrement)
ϕ	1	01	IR + 1	(Increment)
0	1	10	IR(8-15) <- I	(Load MSB)
1	1	10	IR(0-7) <- I	(Load LSB)
ϕ	1	11	0	(Clear)

Figure 2: Graphic symbol of the IR register (Left) and its characteristic table (Right)

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

(Part-2a) Design the system shown in Figure 3 which consists of four 8-bit general purpose registers: **R1**, **R2**, **R3,** and **R4** and four 8-bit temporary registers: **T1**, **T2**, **T3,** and **T4**. The details of inputs and outputs are as follows.

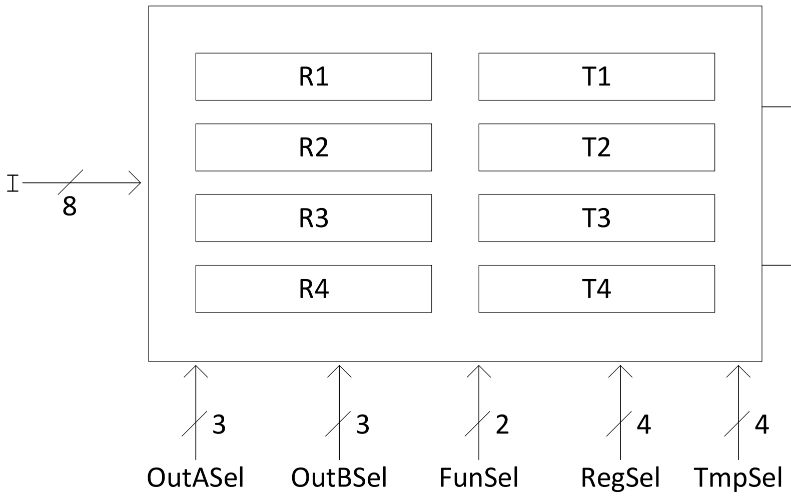


Figure 3: 8-bit general purpose and temporary registers, inputs, and outputs

OutASel and **OutBSel** are used to feed output lines **OutA** and **OutB**, respectively. 8 bits of the selected registers are output to **OutA** and **OutB**. Figure 4 shows selection of output registers based on the **OutASel** and **OutBSel** control inputs.

OutASel	OutA
000	R1
001	R2
010	R3
011	R4
100	T1
101	T2
110	T3
111	T4

OutBSel	OutB
000	R1
001	R2
010	R3
011	R4
100	T1
101	T2
110	T3
111	T4

Figure 4: OutASel and OutBSel controls

RegSel and **TmpSel** are 4-bit signals that select the registers to apply the function that is determined by the 2-bit **FunSel** (*Figure 5*) signal. The selectes register by **Regsel** and **TmpSel** are shown in *Figure 6* and *Figure 7*, respectively.

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

Figure 5: FunSel Control Input

RegSel	Enabled General Purpose Registers
0000	ALL general purpose registers are enabled (Function selected by FunSel will be applied to R1, R2, R3 and R4)
0001	R1, R2 and R3 are enabled (Function selected by FunSel will be applied to R1, R2 and R3)
0010	R1, R2 and R4 are enabled, Function selected by FunSel will be applied to R1, R2 and R4
0011	R1 and R2 are enabled (Function selected by FunSel will be applied to R1 and R2)
0100	R1, R3 and R4 are enabled (Function selected by FunSel will be applied to R1, R3 and R4)
0101	R1 and R3 are enabled (Function selected by FunSel will be applied to R1 and R3)
0110	R1 and R4 are enabled (Function selected by FunSel will be applied to R1 and R4)
0111	Only R1 is enabled (Function selected by FunSel will be applied to R1)
1000	R2, R3 and R4 are enabled (Function selected by FunSel will be applied to R2, R3 and R4)
1001	R2 and R3 are enabled (Function selected by FunSel will be applied to R2 and R3)
1010	R2 and R4 are enabled (Function selected by FunSel will be applied to R2 and R4)
1011	Only R2 is enabled (Function selected by FunSel will be applied to R2)
1100	R3 and R4 are enabled (Function selected by FunSel will be applied to R3 and R4)
1101	Only R3 is enabled (Function selected by FunSel will be applied to R3)
1110	Only R4 is enabled (Function selected by FunSel will be applied to R4)
1111	N0 general purpose register is enabled (All R1, R2, R3 and R4 registers retain their values)

Figure 6: RegSel Control Input

TmpSel	Enabled Temporary Registers
0000	ALL temporary registers are enabled (Function selected by FunSel will be applied to T1, T2, T3 and T4)
0001	T1, T2 and T3 are enabled (Function selected by FunSel will be applied to T1, T2 and T3)
0010	T1, T2 and T4 are enabled, Function selected by FunSel will be applied to T1, T2 and T4
0011	T1 and T2 are enabled (Function selected by FunSel will be applied to T1 and T2)
0100	T1, T3 and T4 are enabled (Function selected by FunSel will be applied to T1, T3 and T4)
0101	T1 and T3 are enabled (Function selected by FunSel will be applied to T1 and T3)
0110	T1 and T4 are enabled (Function selected by FunSel will be applied to T1 and T4)
0111	Only T1 is enabled (Function selected by FunSel will be applied to T1)
1000	T2, T3 and T4 are enabled (Function selected by FunSel will be applied to T2, T3 and T4)
1001	T2 and T3 are enabled (Function selected by FunSel will be applied to T2 and T3)
1010	T2 and T4 are enabled (Function selected by FunSel will be applied to T2 and T4)
1011	Only T2 is enabled (Function selected by FunSel will be applied to T2)
1100	T3 and T4 are enabled (Function selected by FunSel will be applied to T3 and T4)
1101	Only T3 is enabled (Function selected by FunSel will be applied to T3)
1110	Only T4 is enabled (Function selected by FunSel will be applied to T4)
1111	N0 temporary register is enabled (All T1, T2, T3 and T4 registers retain their values)

Figure 7: TmpSel Control Input

For example: If **RegSel** is 1001, **TmpSel** is 1111, and **FunSel** is 01, then the registers **R2** and **R3** will be incremented with next clock cycle. **R1** and **R4** will not be affected since they are not enabled by **RegSel**. Similarly, temporary registers **T1**, **T2**, **T3**, **T4** will not be affected since they are not enabled by **TmpSel**.

(Part-2b) Design the **address register file (ARF)** system shown in Figure 8 which consists of three 8-bit address registers: **program counter (PC), address register (AR), and stack pointer (SP)**. FunSel and Regsel works as in **Part-2a**.

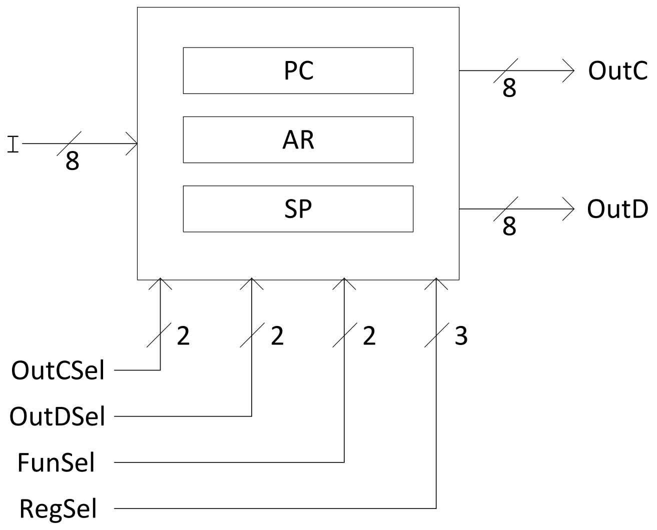


Figure 8: 8-bit address registers, inputs, and outputs

OutCSel and **OutDSel** are used to feed output lines **OutC** and **OutD**, respectively. 8 bits of the selected registers are output to **OutC** and **OutD**. Figure 9 shows selection of output registers based on the **OutCSel** and **OutDSel** control inputs.

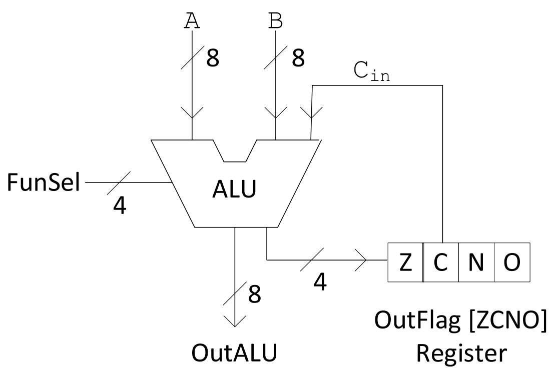
OutCSel	OutC
00	PC
01	PC
10	AR
11	SP

OutDSel	OutD
00	PC
01	PC
10	AR
11	SP

Figure 9: OutCSel and OutDSel controls

(Part-3) Design an Arithmetic Logic Unit (ALU) that has two 8-bit inputs, an 8-bit output, and a 4-bit output for **z**ero, **n**egative, **c**arry, and **o**verflow flags. The ALU is shown on the left side of Figure 10. 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 OutALU) are given on the right side of Figure 10:

- **FunSel** selects the function of the ALU.
- **OutALU** 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 **OutALU** is zero (e.g., when **NOT B** is zero).
- **C (carry)** bit is set if **OutALU** 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!**



FunSel	OutALU	Z	C	N	O
0000	A	√	–	√	–
0001	B	√	–	√	–
0010	NOT A	√	–	√	–
0011	NOT B	√	–	√	–
0100	A + B	√	√	√	√
0101	A + B + Carry	√	√	√	√
0110	A - B	√	√	√	√
0111	A AND B	√	–	√	–
1000	A OR B	√	–	√	–
1001	A XOR B	√	–	√	–
1010	LSL A	√	√	√	–
1011	LSR A	√	√	√	–
1100	ASL A	√	–	√	√
1101	ASR A	√	–	–	–
1110	CSL A	√	√	√	√
1111	CSR A	√	√	√	√

Figure 10: The ALU (Left) and its characteristic table (Right)

(**C**ircular | **A**rithmetic | **L**ogical) **S**hift (**L**eft | **R**ight) operations are depicted in *Figure 11*, *Figure 12, and Figure 13*.

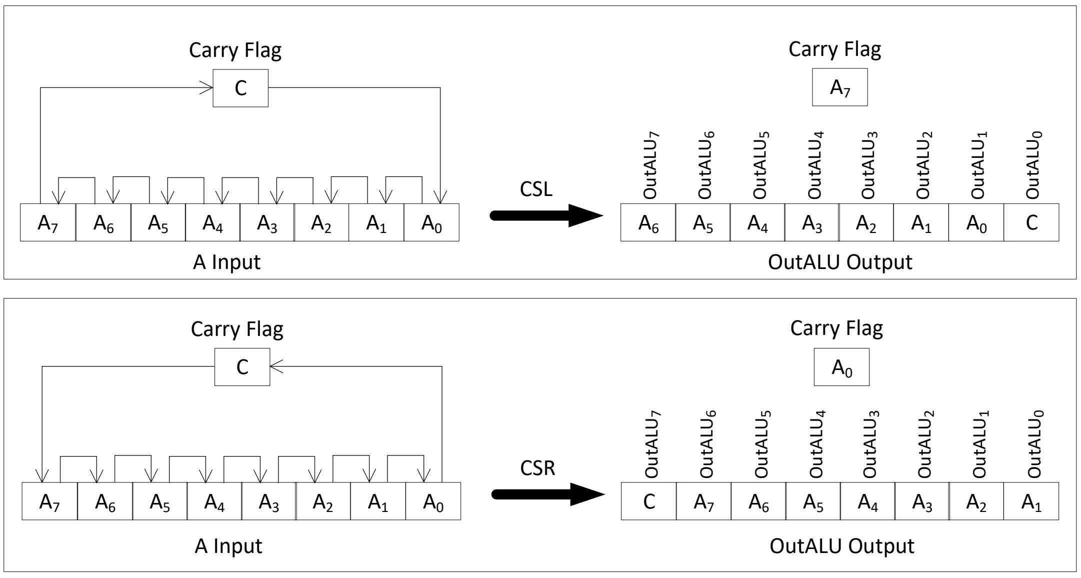

Carry Flag
C
A$_7$ A$_6$ A$_5$ A$_4$ A$_3$ A$_2$ A$_1$ A$_0$
A Input
CSL
Carry Flag
A$_7$
OutALU$_7$ OutALU$_6$ OutALU$_5$ OutALU$_4$ OutALU$_3$ OutALU$_2$ OutALU$_1$ OutALU$_0$
A$_6$ A$_5$ A$_4$ A$_3$ A$_2$ A$_1$ A$_0$ C
OutALU Output
Carry Flag
C
A$_7$ A$_6$ A$_5$ A$_4$ A$_3$ A$_2$ A$_1$ A$_0$
A Input
CSR
Carry Flag
A$_0$
OutALU$_7$ OutALU$_6$ OutALU$_5$ OutALU$_4$ OutALU$_3$ OutALU$_2$ OutALU$_1$ OutALU$_0$
C A$_7$ A$_6$ A$_5$ A$_4$ A$_3$ A$_2$ A$_1$
OutALU Output

Figure 11: Circular Shift Operations

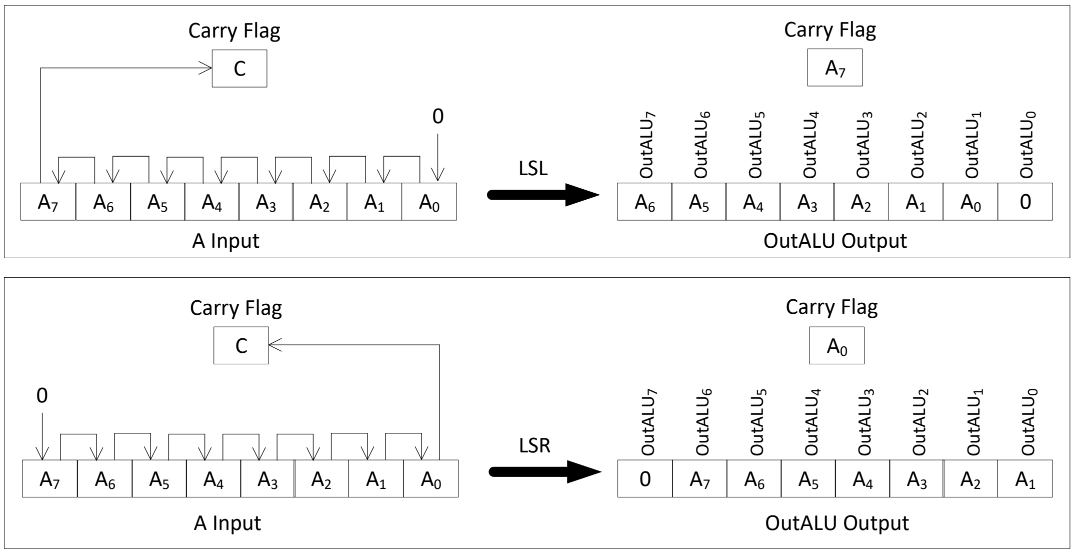

Carry Flag
C
0
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
A Input
LSL
Carry Flag
A_7
$OutALU_7$ $OutALU_6$ $OutALU_5$ $OutALU_4$ $OutALU_3$ $OutALU_2$ $OutALU_1$ $OutALU_0$
A_6 A_5 A_4 A_3 A_2 A_1 A_0 0
OutALU Output
Carry Flag
C
0
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
A Input
LSR
Carry Flag
A_0
$OutALU_7$ $OutALU_6$ $OutALU_5$ $OutALU_4$ $OutALU_3$ $OutALU_2$ $OutALU_1$ $OutALU_0$
0 A_7 A_6 A_5 A_4 A_3 A_2 A_1
OutALU Output

Figure 12: Logical Shift Operations

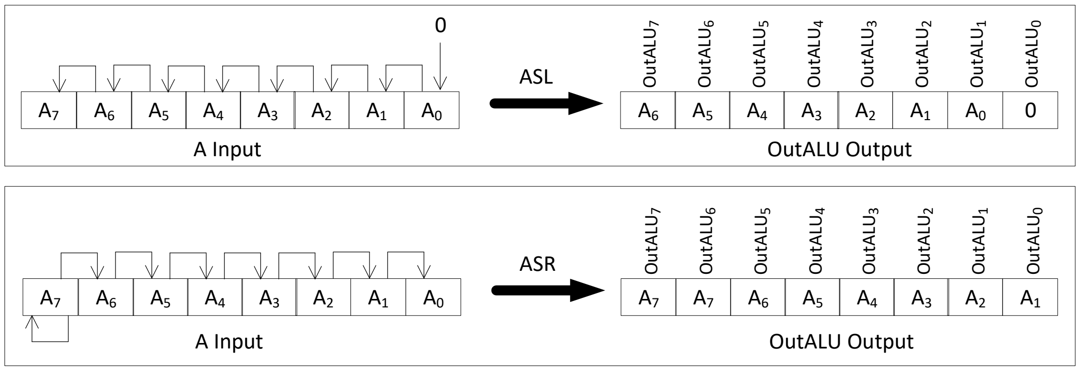
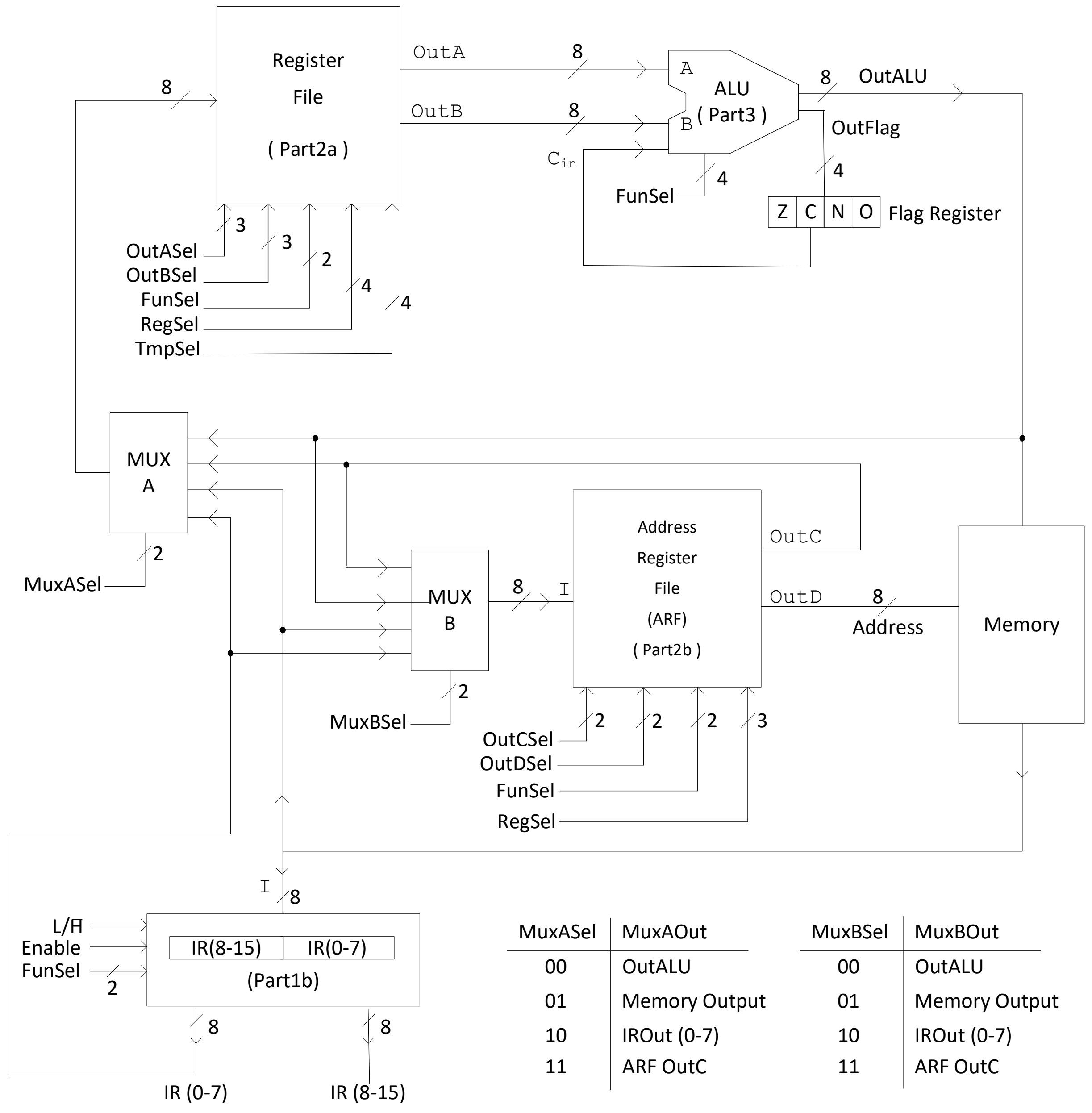

0
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
A Input
ASL
$OutALU_7$ $OutALU_6$ $OutALU_5$ $OutALU_4$ $OutALU_3$ $OutALU_2$ $OutALU_1$ $OutALU_0$
A_6 A_5 A_4 A_3 A_2 A_1 A_0 0
OutALU Output
A_7 A_6 A_5 A_4 A_3 A_2 A_1 A_0
A Input
ASR
$OutALU_7$ $OutALU_6$ $OutALU_5$ $OutALU_4$ $OutALU_3$ $OutALU_2$ $OutALU_1$ $OutALU_0$
A_7 A_7 A_6 A_5 A_4 A_3 A_2 A_1
OutALU Output

Figure 13: Arithmetic Shift Operations

(Part-4) Implement the organization in Figure 14. Please note that, **the whole system uses the same single clock.**



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

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

Figure 14: ALU System

Submission:

Implement your design in logisim software, 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**). This compressed file should contain your design files (.circ) and a report that contains:

- the number&names of the students in the group
- list of control inputs and corresponding functions for your design

Group work is expected for this project. All the 3 student members of the group **must** design together. Make sure to connect pins (under Wiring group of logisim) to the inputs and control inputs of your design, so that different inputs and functions can be tested. Similarly connect your inputs and outputs to a “Hex Digit Display" in logisim (under Input/output group of logisim) so that the test outputs can be observed and use proper labelling to improve the clarity of your circuits.