Q1.

A - ⓐ) Since the instruction of 6bit is too short to get all the information in the instruction like RISC-V, I decided to set some registers X5, X6, X7, and X8 to be used in certain purpose when certain instruction executed, otherwise it can be use as usual register. Also set X0 have fixed value, X0=0. -

X0	X1	X2	X3	X4	X5 (save data)	X6 (blt compare)	X7 (blt value)	X8 (now)
0	U	U	U	U	U	U	U	U

The instruction set I made are just like the table below. Since all the instruction consist of only 6 bits, I assigned 2~3 bits to opcode. To get large immediate value, opcode of immediate instruction is 2 bits so that it can get 4bit value. Opcode of other instructions are not starting with 00 to distinguish. Also, I made put and get instructions to copy the value of X8 to other register, and vice versa.

Instruction	2bits	1bit	3bits	Comments
Imm	00	4bit immediate		4bit number (0 to 15) saved in X8 (now register)
Get	01	0	rs	Get value of rs to X8.
Put	01	1	rd	put the value of X8(now register) to the rd register
Load	10	0	rs	Load data from memory with mem address (rs value+x8 value) to X8(now register)
Save	10	1	rs	Save data at X5 (save data register) to memory in the address of (memory address value in X8(now register) + rs value)
Add	11	0	rs	X8(now register) = X8(now register) + (value of rs register)
blt	11	1	000	If the value of X6 (blt compare) is less than X7 (blt value), go back to instruction of (current pc value – (x8 value)).

Then the code given can be written as follows. Since all the operands needed are in the registers, I assumed that X2 value is address off A[0], X1 has the address of i, which has initialized value 0, and X3 has address of 100.

#	Instruction	Machine code	X8 (now) value	Comments
1	Imm 0	000000	0	X8=0
2	Load x1	100001	0	X8=0
3	Put x6	011110	0	X6=0
4	Load x3	100011	100	X8=100
5	Put x7	011111	100	X7=100
6	imm 2	000010	2	X8=2
7	put x3	011011	2	X3=2
8	get x6	010110	i	X8=i
9	load x2	100010	i	X8=(A[0] address + i) data=A[i] data
10	add x3	110011	A[i] data + 2	X8=A[i] data + 2
11	put x5	011101	A[i] data + 2	X5=A[i] data + 2
12	imm 1	000001	1	X8=1
13	add x6	110110	i+1	X8=i+1
14	put x6	011110	i+1	X6=i+1
15	Save x2	101010	i+1	Save x5 value to memory of address (A[0] address+x8(i+1)=A[i+1]
16	Imm 9	001001	9	X8=9
17	Blt	111000	9	If x6<x7, go back to the instruction (current pc value) - (9) =instruction #8. If not, current pc value just increases.

Then the dynamic instruction count to execute the given code sequence would be 7+10*100=1007.

A-ⓑ) To make 8-bit instruction, I decided to use some of the registers in certain purpose to reduce the register numbers in a instruction. The following table shows the register purpose and fixed values.

X0	X1	X2	X3	X4	X5 (save data)	X6 (blt compare)	X7 (blt value)	X8 (now)
0	U	U	U	U	U	U	U	U

Then the 8-bit instruction set architecture I made to execute the given code is like below. Like the 6-bit ISA, I put get and put instructions to use 'now' register, X8. Also, I added addi and puti instructions to make some process simple.

Instruction	2bits	1bit	1bit	1bit	3bits	Comments
add	00	rs1			rs2	add value of rs1 and rs2 and put the result in rs2.
imm	01	0	5bit immediate			Get the immediate value at x8(now register)
get	01	1	2bit immediate		rs	X8 get the value of (rs value+2bit immediate)
blt	10	0	5bit immediate			If value of X6 (blt compare) is less than X7 (blt value), go back to instruction of (current pc value – (immediate value)).
save	10	1	0	0	rs	Save data in X5(save data register) to the memory of the address of (x8 value + rs value)
load	10	1	0	1	rs	Load data in X8(now register) from the memory of the address rs+x8.
put	10	1	1	1	rd	put the value of x8 in rd register
addi	11	0	2bit immediate		rs	Add value of 2 bit immediate to rs value and put the result in rs.
puti	11	1	2bit immediate		rd	Put immediate value to rd

Then the c-code given can be expressed in the 8bit ISA as follows. I assumed that address of A[0] is in X2, x4 has address of i, which has initialized value 0, and x3 has address of 100.

#	Instruction	Machine code	X8 (now) value	Comments
1	Imm 0	01000000	0	X8=0
2	Load x4	10101100	0	X8=i=0
3	Put x6	10111110	0	X6=i=0
4	Load x3	10101011	100	X8=100
5	Put x7	10111111	100	X7=100
6	Get 0 x6	01100110	i	X8=i
7	load x2	10101010	A[i] data	X8=(A[0]+i) data=A[i] data
8	put x5	10111101	A[i] data	X5=A[i] data
9	Addi 2 x5	11010101	A[i] data	X5=A[i] data+2
10	Addi 1 x6	11001110	A[i] data	X6=i+1
11	save x2	10100010	i+1	Save x5 data to memory address of A[0] + i+1=A[i+1]
12	blt 6	10000110	i+1	If X6 value < X7 value, go to the instruction (current pc value)-6=instruction #6

Then the dynamic instruction count to execute the given code sequence would be 5+7*100=705. The count is much less than the preceding 6-bits instruction ISA. Since there are more available bits in instruction to use, it was possible to use various instruction types with longer opcode, with large immediate value, and with less instruction to execute. However, there is no 2 bits for sanity check, which makes the ISA vulnerable to error, and various length of opcode requires control unit to read more opcode bits.

B) Although the address system has changed, the relative address value does not change since address value is assigned for every block bits (8bits and 6bits each). Also, since there is no mention about any change in resistor size, assigning 100 to register x7 would not be a problem. Therefore, without modification, the instruction works fine in changed condition.

C) Both ISAs can be used to convert the given code, since multiplication is repetition of addition. By using blt instruction in both cases, multiplication can be implemented, and each assembly codes are just like follows.

In 6-bit ISA, I assumed that address of x and y are in register x1 and x2.

#	Instruction	Machine code	X8 (now) value	Comments
1	Imm 4	000100	4	X8=4
2	Put x5	011101	4	X5=4
3	Imm 0	000000	0	X8=0
4	Save x1	101001	0	Save x5=4 in memory x
5	Imm 3	000011	3	X8=3
6	Put x5	011101	3	X5=3
7	Imm 0	000000	0	X8=0
8	Save x2	101010	0	Save x5=3 in memory y
9	Load x1	100001	4	Load memory x value 4 in x8
10	Put x3	011011	4	X3=4
11	Load x2	100010	3	Load memory y value 3 in x8
12	Put x7	011111	3	X7=3
13	Imm 0	000000	0	X8=0
14	Put x6	011110	0	X6=0
15	Put x4	011100	0	X4=0
16	Get x4	010100	4*i	X8=4*i
17	Add x3	110011	4*(i+1)	X8=x8+4=4*(i+1)
18	Put x4	011100	4*(i+1)	X4=4(i+1)
19	Imm 1	000001	1	X8=1
20	add x6	110110	i+1	X8=i=i+1
21	Put x6	011110	X6+1	X6=x6+1
22	Imm 7	000111	7	X8=7
23	Blt	111000	7	If X6 value < X7 value, go to the instruction (current pc value)-7=instruction #16
24	Get x4	010100	12	X4=(2+1)*4=12
25	Put x5	011101	12	X5=12
26	Imm 0	000000	0	X8=0
27	Save x1	101001	0	Save x5=12 value in memory x.

Also, in 8-bit ISA, I assumed that address of x and y are in x1 and x2.

#	Instruction	Machine code	X8 (now) value	Comments
1	Puti 4 x5	11111101	4	X5=4
2	Imm 0	01000000	0	X8=0
3	Save x1	10100001	0	Save 4 at x
4	Puti 3 x5	11111101	3	X5=3
5	Imm 0	01000000	0	X8=0
6	Save x2	10100010	0	Save x5=3 at y

7	Imm 0	01000000	0	X8=0
8	Load x1	10101001	4	Load 4 from x
9	Put x3	10111011	4	X3=4
10	Imm 0	01000000	0	0
11	Put x6	10111110	0	X6=0
12	Put x4	10111100	0	X4=0
13	Load x2	10101010	3	Load 3 from y
14	Put x7	10111111	3	X7=3
15	Add x3 x4	00011100	3	X4=x4+4=(i+1)*4
16	Addi 1 x6	11001110	3	X6=i=i+1
17	Blt 2	10000010	3	If X6 value < X7 value, go to the instruction (current pc value)-2=instruction #15
18	Get 0 x4	01100100	12	X8=12
19	Put x5	10111101	12	X5=12
20	Imm 0	01000000	0	X8=0
21	Save x1	10100001	0	Save x5=12 at x

Therefore, both cases can convert the given c code into the assembly code without modification.

D) For the 6-bit instruction, there can be no improvement since the reason of long instruction flow is that there are no enough bits for more information, like bits for assigning more registers in instruction. The instructions set architecture allocates its limited bits to most essential part, opcode and one or two operands.

However, for the 8-bit instruction, there are too many 'Imm 0' before save to get the right memory address to save in instruction sets for code 2. Therefore, we can add save0 instruction with opcode 10110, which saves data in x5 to the memory address in rs register. It is instruction which is same as 'imm 0' and 'save' instruction combined.

Instruction	2bits	1bit	1bit	1bit	3bits	Comments
add	00	rs1			rs2	add value of rs1 and rs2 and put the result in rs2.
imm	01	0	5bit immediate			Get the immediate value at x8(now register)
get	01	1	2bit immediate		rs	X8 get the value of (rs value+2bit immediate)
blt	10	0	5bit immediate			If value of X6 (blt compare) is less than X7 (blt value), go back to instruction of (current pc value – (immediate value)).
save	10	1	0	0	rs	Save data in X5(save data register) to the memory of the address of (x8 value + rs value)
Save0	10	1	1	0	rs	Save data in X5(save data) to the memory of the address of rs value
load	10	1	0	1	rs	Load data in X8(now register) from the memory of the address rs+x8.
put	10	1	1	1	rd	put the value of x8 in rd register
addi	11	0	2bit immediate		rs	Add value of 2 bit immediate to rs value and put the result in rs.
puti	11	1	2bit immediate		rd	Put immediate value to rd

By using this instruction, we can reduce 3 instructions for code2, by combining #3 and #4, #7 and #8, #22 and #23.

Those two ISAs (6bit, 8bit) have almost fixed instruction fields, which make the whole operation simple. Also, in limited instruction length they get instruction operands as many as possible, while cover the shortage with now register, which do not use in instruction field. In D, some instructions are added which can replace instructions that commonly used together so that reduce the instruction flow. Therefore, I think those designs are my best in given conditions.

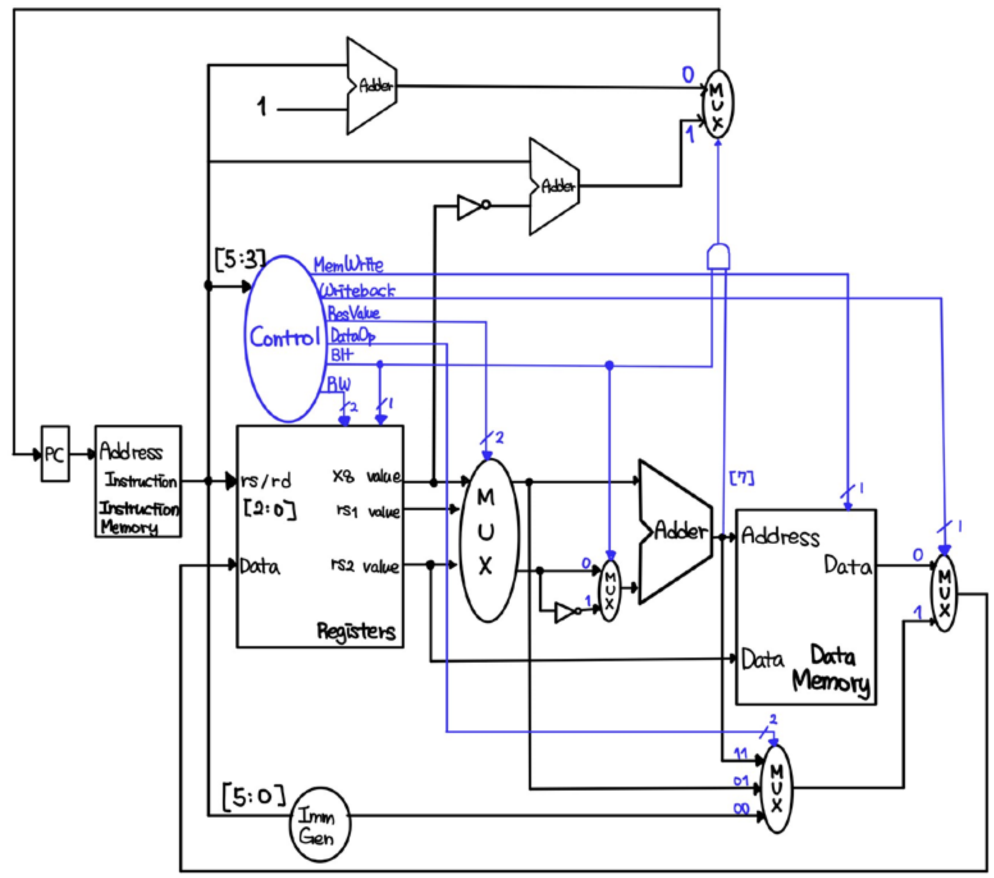
Q2.

A-①) 3 adders, 6bit

I implemented hardware design like the figure, defining some control signals. BLT signal controls Register to get X6 and X7 value to compare their values, and a MUX since the comparing operation uses subtraction. It controls PC value calculation with the not-main adder-output [7], since it gets 1 when X6<X7. Each control signals operates like the control signal table, and since some operations cannot be shown on the figure, I put another table to make their operations clear.

	Imm(00)	Get(010)	Put(011)	Load(100)	Save(101)	Add(110)	Blt
Blt	0	0	0	0	0	0	1
MemWrite	0	0	0	0	1	0	0
WriteBack	1	1	1	0	xx	1	xx
ResValue	xx	00	01	01	01	01	11
DataOp	00	01	01	xx	xx	11	xx
RW	10	10	11	10	00	10	00

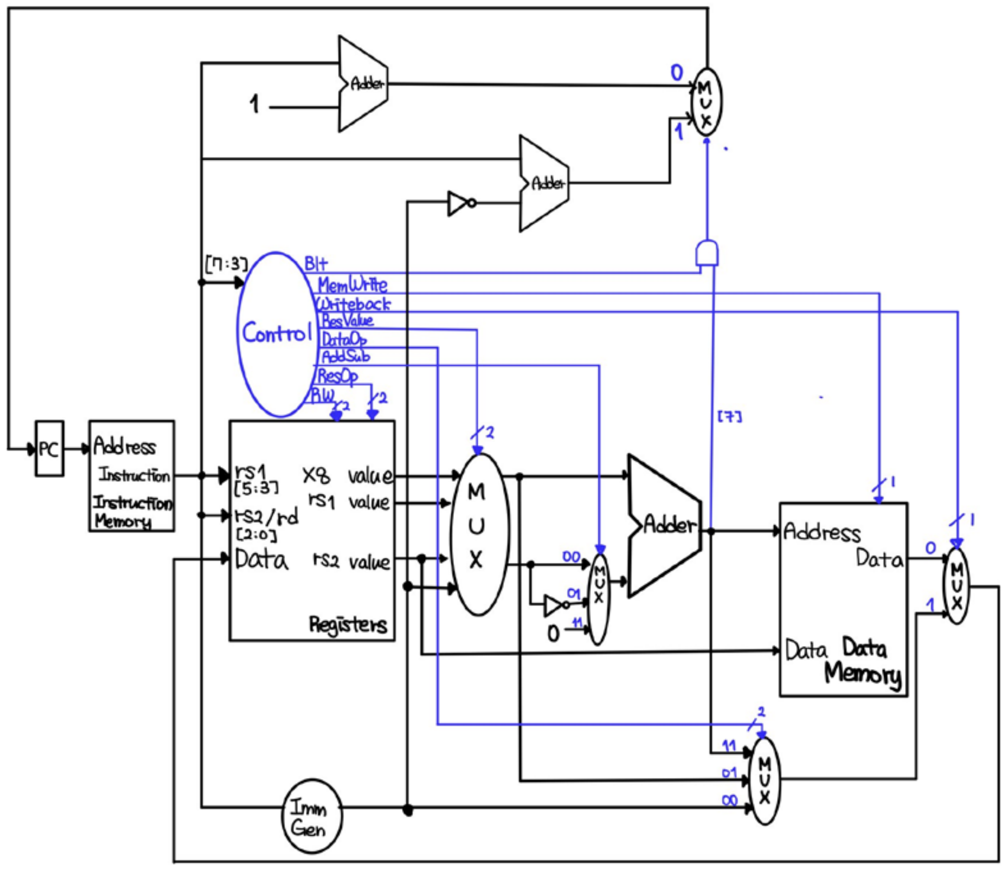
BLT	Register Output		ResValue	MUX Output		RW	Write Register
	Rs1 value	RS2 value		DataOp MUX	BLT MUX		
0	RS	X5	00	RS1 data	X8	00	No write
1	X6	X7	01	X8	RS1 data	10	X8
			11	RS1 data	RS2 data	11	Rs/rd



In the ISA, X8, now register, is frequently used, therefore the register value is always read. By using now register, there are three register data outputs, so I put MUX to choose two among them. Also, since the instruction can save immediate value, register value, and result of adder-using-operation, I put another MUX to choose which one to written in register. Imm_gen part just get the immediate value of the instruction and return the same value in 8 bit. Since the design has minimum control signal, appropriate register design, and appreeopriate use of MUXs, I think it is quite good.

A-②) 3 adders, 8 bit

Just as A-①, there are two tables for conrol signals and the operation that cannot be shown by the design figure.



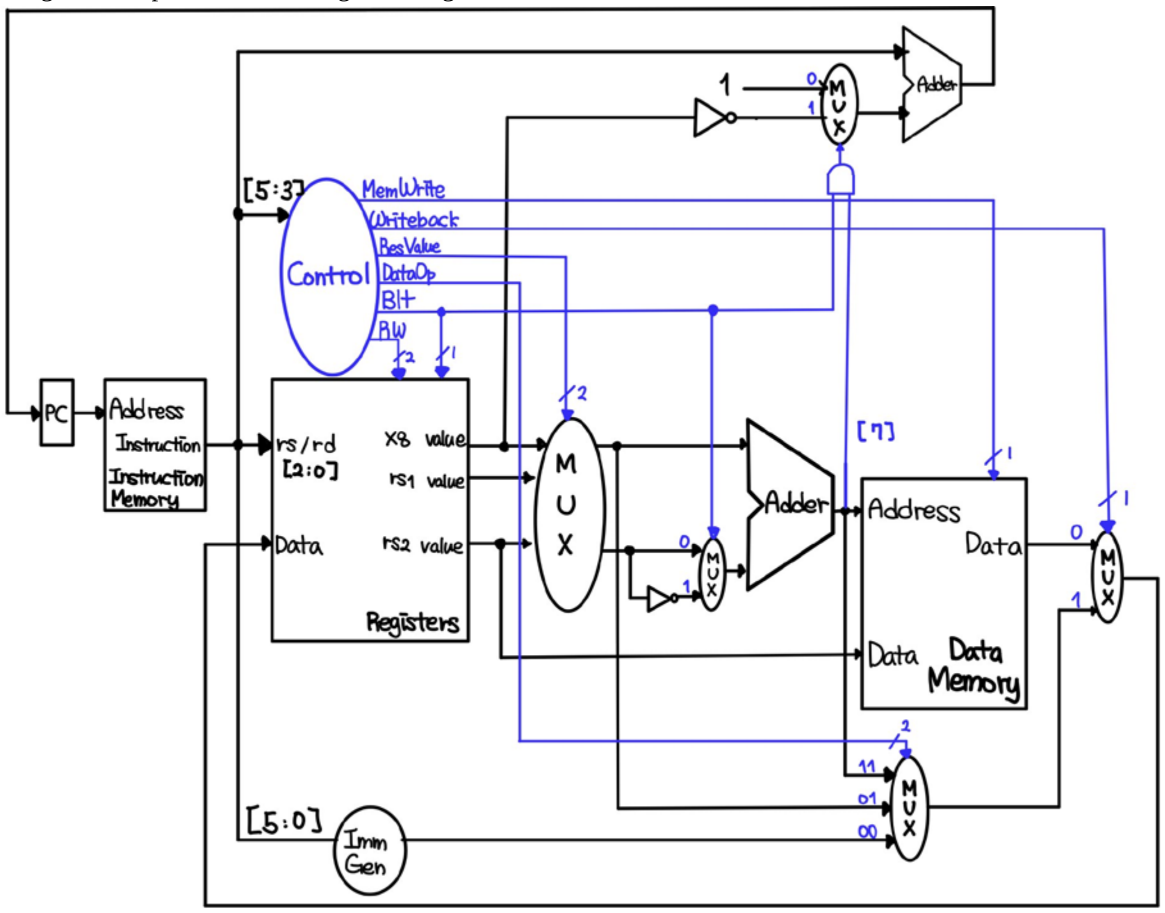
	add	Imm	Get	Blt	Save	Save0	Load	Put	Addi	puti
Blt	0	0	0	1	0	0	0	0	0	0
MemWrite	0	0	0	0	1	1	0	0	0	0
WriteBack	1	1	1	X	X	X	0	1	1	1
ResValue	11	Xx	00	11	01	xx	01	Xx	00	Xx
ResOp	01	Xx	01	11	00	Xx	01	xx	01	xx
DataOp	11	00	01	Xx	Xx	Xx	Xx	01	11	00
RW	11	10	10	00	00	00	10	11	11	11
AddSub	00	x	x	01	00	11	00	xx	00	xx

ResOp	Register Output		ResValue	MUX Output		RW	Write Register
	Rs1 value	RS2 value	DataOp MUX	BLT MUX	RS2 value		
00	RS2	X5	00	RS2 data	Imm	00	No write
01	RS1	RS2	01	X8	RS2 data	10	X8
11	X6	X7	11	RS1 data	RS2 data	11	Rs2/rd

Implementing save0 required register value as a address in memory, therefore I put 0 input to MUX to get the value, and added a control signal for it. The design uses control signal appropriately, get register values and use them well despite the instruction length limit by using always-on register value x8 with appropiate register design. Since the design supports instruction in limited condition very well, I think it is good.

B - ①) 2 Adders, 6bit

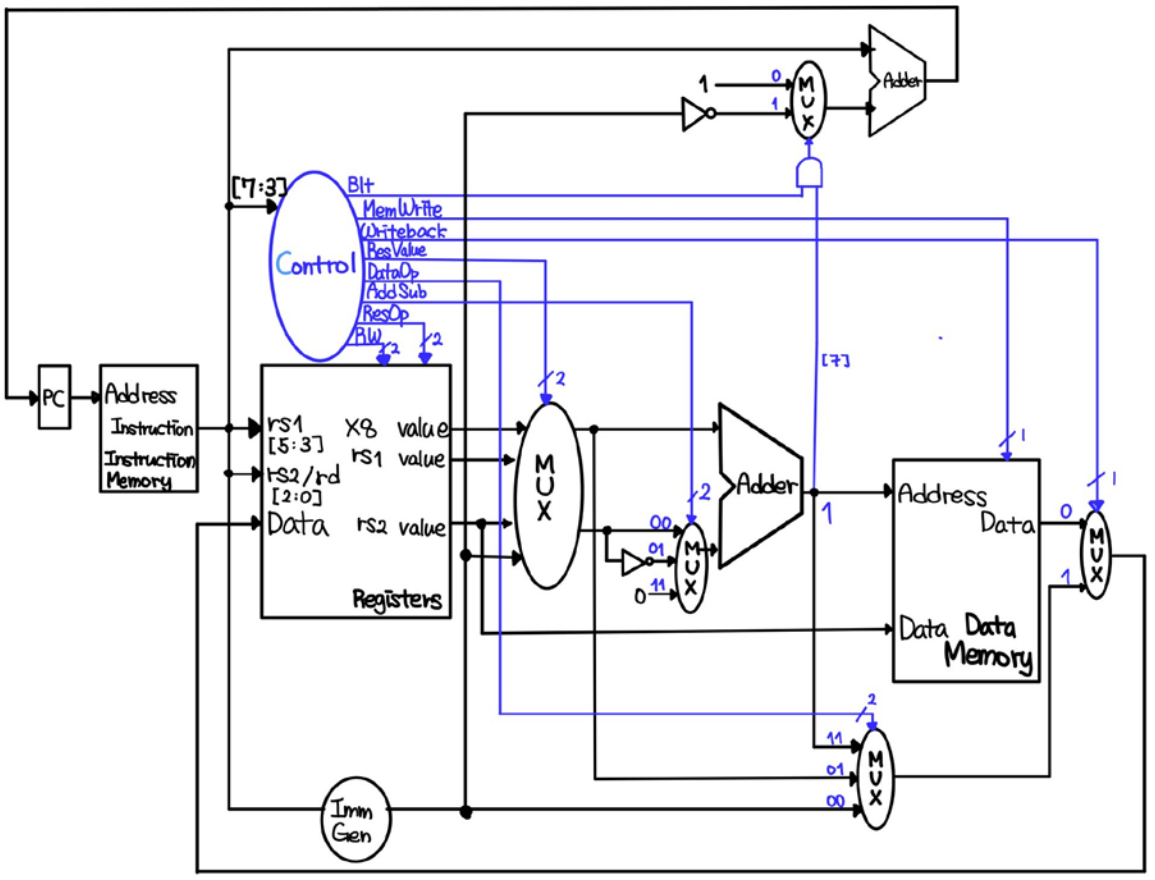
To reduce one adder, I changed the order of MUX and adder which are used for calculating PC values. Since the design is for same ISA, control signal and operation according to the signal is same as A - ①.



By using one control signal for 3 different operation, I could reduce the number of signals. Also, using MUXs simplified datapath, which was complicated because of the operations that requires fixed registers, different registers or immediate values. I think this is the best design in limitation of insturction length and two adders.

B - ②) 2 Adders, 8bit

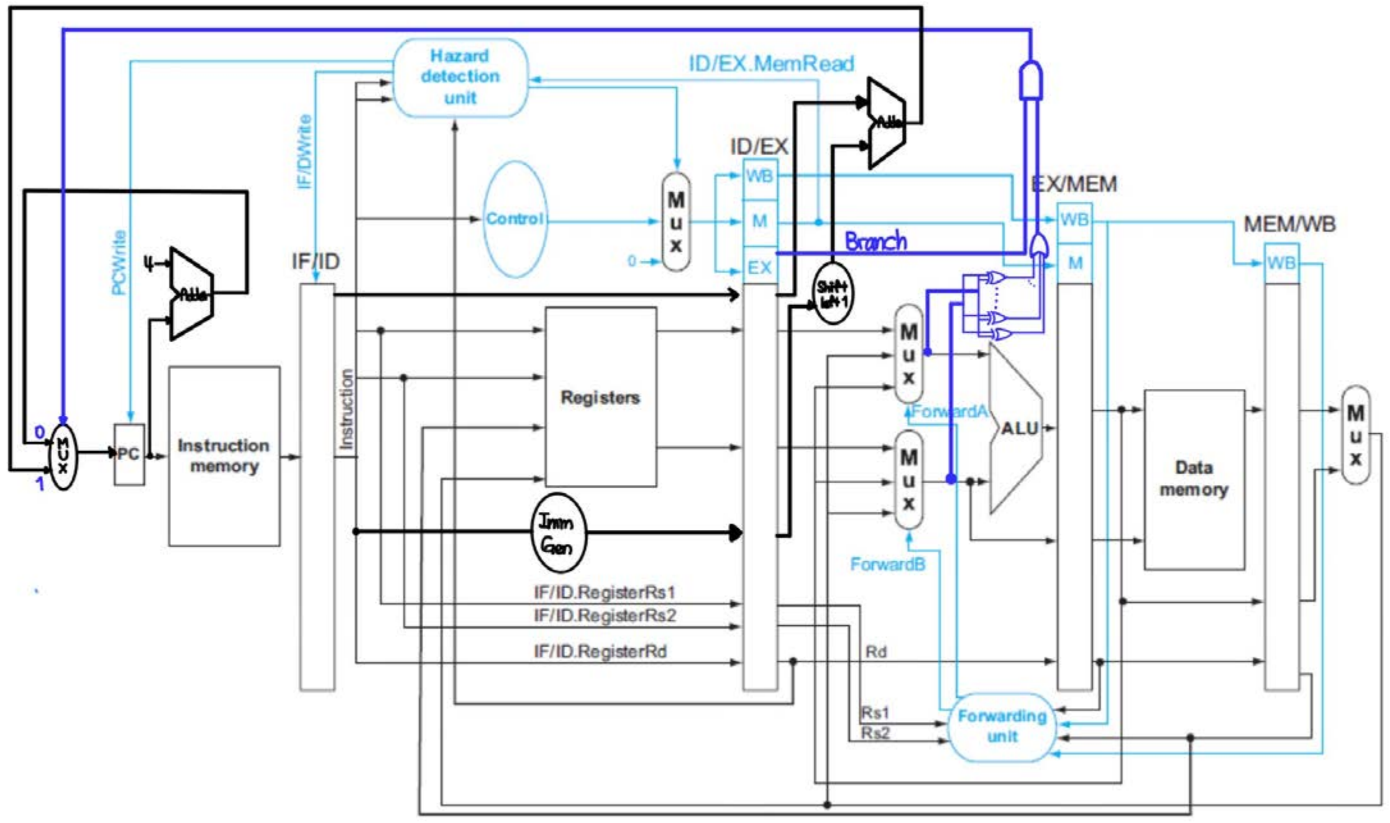
To reduce one adder, I changed the order of MUX and adder which are used for calculating PC values, just like the previous one. The design has same control signal system and opration according to the signal as A - ②.



Though there is no difference without the blt part, I think the design is good since it its already has reduced signals and datapath by using MUXs. Though it has various length of opcode, register number in instructions are in fixed location, registers values are well chosen and used.

Q3.

A) There should be adder, imm_gen, shift left 1 part for ID stage since all these operations should occur in it. Also, it needs control signal for branch instruction MUX before the PC. The signal is result of AND operation about branch control signal and True/False signal for whether the registers have same value. The part to detect if the values are equal or not is consisted of parallel XORs for every bit and OR part which get the results of preceding XOR operations.

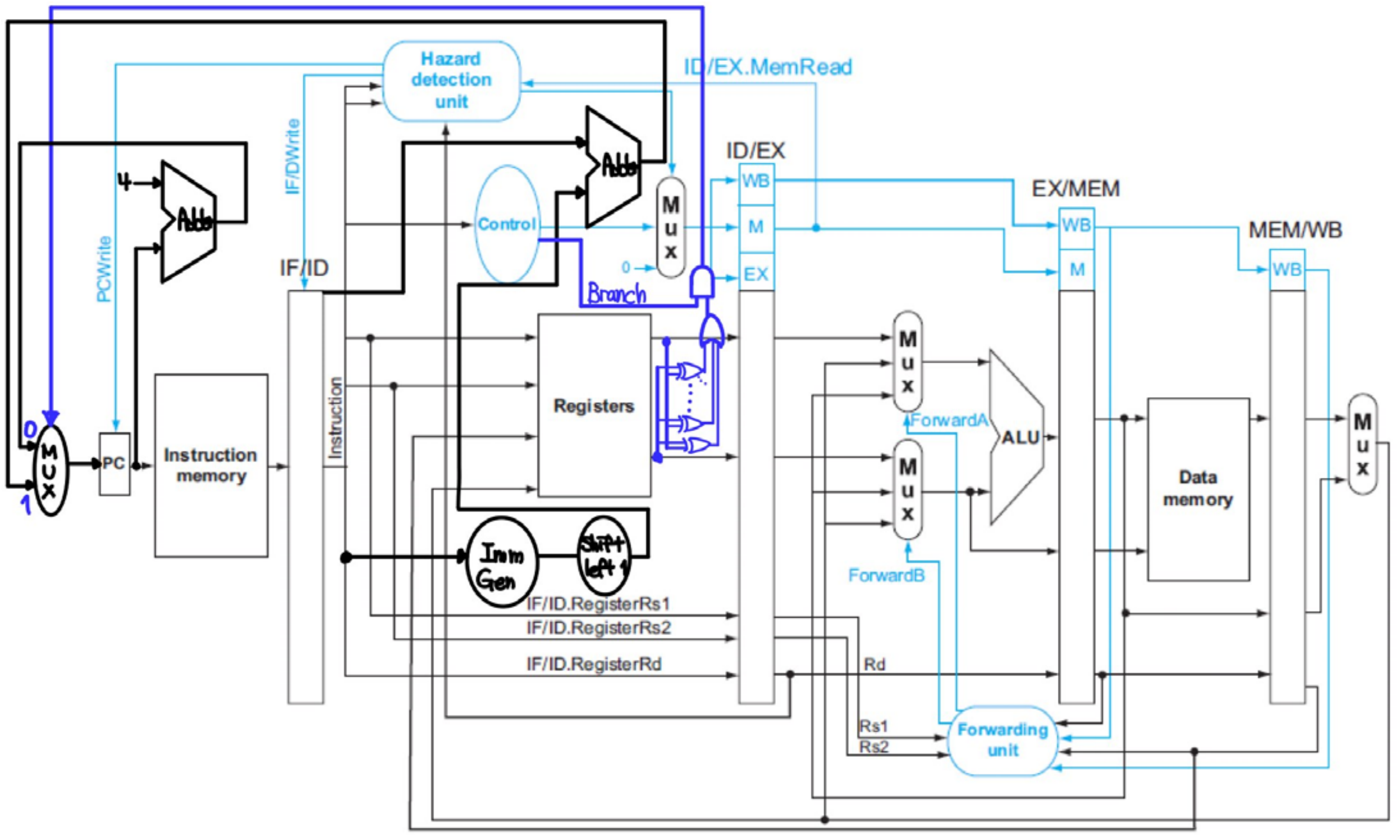


When the one-cycle preceding instruction is not loading instruction, there will be no hazard since the forwarding unit for EX stage already exists. However, the hardware design shows data hazard when the loading instruction is just before the beq instruction, just like the table. Since the getting data from memory occurs at the end of the MEM stage while the data is needed when the EX stage starts. Therefore, there should be a bubble between the loading instruction and beq instruction, and it requires another control signal which delays the instruction execution and makes bubble.

#	instruction
…	
32	ld X2, 12(x5)
36	beq x2 x3 32
…	

B)

The design also needs adder, imm_gen, shift left 1 part, and part for control signal to branch instruction MUX before the PC, consisted of XOR-OR part for equality test, and AND operation to decide whether the branch takes or not. s



The hardware design shows data hazard when the instruction like the first table executed, since there is no forwarding unit for ID stage. Register X2 should be written before the beq instruction executed. However, in pipeline system, the register could not be written before the branch instruction gets EX stage. Therefore, there should be another forwarding unit and hazard detection unit to decide if the XOR-OR part should get bypassed data, since the existing units are just for EX stage operation.

Even if we put those new forwarding unit and hazard detection unit, there still can be data hazard which make bubble necessary, when ALU operation and loading operation executed just before the branch operation, just like second and third table. For the second table, there should be one bubble between the add and beq instructions since the ALU calculation for add operation could not be finished before the beq operation starts. For the third table, since loading instruction gets the memory value at the end of the MEM stage while the value is needed for ID stage of beq instruction, there should be two bubbles for the intended operation.

#	instruction
…	
32	add x2 x4 x5
36	sub x8 x6 x1
40	beq x2 x3 32
…	

#	instruction
…	
32	add x2 x4 x5
36	beq x2 x3 32
…	

#	instruction
…	
32	ld X2, 12(x5)
36	beq x2 x3 32
…	

<Reference>

David A. Patterson, John L. Hennessy, Computer Organization And Design RISC-V Edition, Morgan Kaufmann.