# CS2323: Computer Architecture, Autumn 2024
## Homework-1: RISC-V Assembly and Binary Numbers
Ahmik Virani - ES22BTECH11001

Q1)

(a) addi x8, x5, -5 //add the immediate value -5 from register x5 and store in register x8

(b) slli x5, x3, 3 //We are multiplying by 8, so left shift the immediate value by 3, that is multiplied by 2^3=8

(c) add x19, x19, x10 //Add the value of register x10 to the value in the register x19 and store it in x19 itself

(d) addi x15, x15, 1 //Add the immediate value of 1 to the value in register x15 and store it in x15 itself

(e) srai x9, x15, 2 //We are dividing by 4, so right shift the value in register x15 by 2, that is divide it by 2^2=4 and store it in x9, but we use srai and not slai to preserve the sign, if negative the division should also be negative, else positive.

(f) addi x12, x0, 24 //x0 return the value 0, so add the immediate value of 24 to 0 and store it in x12

Q2)

(a)
```
ld x9, 160(x5)      //Load M[20] into x9
addi x10, x9, 100   //Compute M[20] + 100
sd x10, 96(x5)      //Store to M[12]
```
(b)
```
ld x9, 160(x5)      //Load M[20] into x9
addi x10, x9, 1     //Compute M[20] + 1
sd x10, 160(x5)     //Store to M[20]
```
(c)
```
ld x9, 40(x5)       //Load M[5] into x9
ld x10, 96(x5)      //Load M[12] into x10
sd x10, 40(x5)      //Store initial value of M[12] to M[5]
sd x9, 96(x5)       //Store initial value of M[5] to M[12]
```
(d)
```
ld x9, 32(x5)       //Load M[4] into x9
slli x9, x9, 32     //Left shift by 32 bits
srli x9, x9, 32     //Now right shift, since logical shifting zeros will come
sd x9, 32(x5)       //Now store this required value back to M[4]
```
(e)
```
ld x9, 16(x5)       //Load M[2] into x9
ld x10, 16(x5)      //Load M[2] into x10
slli x9, x9, 32     //Move the Least significant bits to Most significant bits location
srli x10, x10, 32   //Move the Most Significant bits to least significant bits location
add, x11, x10, x9   //Store the added new flipped value in register x11
sd x11, 16(x5)      //Store required value back to M[2]
```

Q3)

(a) +23

Since it is positive we can directly convert it into binary
Keep dividing by 2 and store the remainder
23 % 2 = 1
11 % 2 = 1
5 % 2 = 1
2 % 2 = 0
1

In binary 23 is 10111 or 0001 0111

(b) -1 is 1111 1111 in binary
+1 is 0000 0001 i.e. just 1%2 = 1 and final value is 0
Next 1's complement -> 1111 1110
Next 2's Complement -> 1111 1111

(c) +255 is not possible to write in 8 bits signed format because if we write 1111 1111 it is interpreted as -1 and not 255

(d) We know that to represent negative numbers we use 2's complement, so 128 in binary is 1000 0000 (using just unsigned, this step is just to create -128)
Converting this to 1's complement -> 0111 1111
2's complement -> 1000 0000
-128 in binary is 1000 0000
2's complement of this is again gives 1000 0000

1's complement of -128 -> 0111 1111
2's complement -> 1000 0000

Note that the range of values in 8 bit binary signed representation is from [-128, 127] therefore as per definition, 2's complement of -128 should be 128, but as we are using 8 bit system, 128 could not be represented

Q4)

(a) 1101 0100 we take 2's complement of this to convert it back into the original number
1's complement -> 0010 1011
2's complement -> 0010 1100
Writing it in decimal -> 2^2 + 2^3 + 2^5 = 44
The original number was -44

(b) 0010 1011
Most Significant bit is 0, so positive
Writing it in decimal -> 2^0 + 2^1 + 2^3 + 2^5 = 43

(c) 1111 1110
1's complement -> 0000 0001
2's complement -> 0000 0010
Decimal -> 2^1 = 2
The original number was -2