CS2323 : HOMEWORK-3

CS2323: Home Work - 3

1. a. addi x15, x22, -45.

Imm	rs1	funct3	rd	opcode
12	5	3	5	7

-45 22 0 15 0010011

1111 1101 0011 10110 000 01111 0010011

1111 1101 0011 1011 0000 0111 1001 0011

0xFD3B0793

b. and x23, x8, x9.

funct7	rs2	rs1	funct3	rd	opcode
7	5	5	3	5	7

0 9 8 7 23 0110011

0000000 01001 01000 111 10111 0110011

0000 0000 1001 0100 0111 1011 1011 0011

0x00947BB3

c. blt x2, x11, 240

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	Opcode
1	6	5	5	3	4	1	7

opcode = 1100011

funct3 = $(4)_{10} = (100)_2$

imm = $(240)_{10} = (00000\ 1111\ 0000)_2$

rs2 = $(11)_{10} = (01011)_2$

rs1 = $(2)_{10} = (00010)_2$

So, finally

0 000111 01011 00010 100 1000 0 1100011

0000 1110 1011 0001 0100 1000 0110 0011

0x0EB14863

d. sd x19, -54(x1)

imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode
7	5	5	3	5	7

opcode = 0100011

funct3 = $(3)_{10} = (011)_2$

rs2 = $(19)_{10} = (10011)_2$

rs1 = $(1)_{10} = (00001)_2$

imm = $(-54)_{10} = (1111\ 1100\ 1010)_2$.

So, finally.

1111110 10011 00001 011 01010 0100011

1111 1101 0011 0000 1011 0101 0010 0011

0xFD30B523

1.e jal x3, -10116

Imm[20]	Imm[10:1]	imm[11]	imm[19:12]	rd	opcode
1	10	1	8	5	7

Opcode is 1101111

rd = $(3)_{10} = (00011)_2$

imm = $(1111\ 111\ 011\ 0000\ 1111100)_2$

20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

[Found using 2's complement and extending signed bit]

$\underline{1}$ $\underline{0000111}\,\underline{110}$ $\underline{1}$ $\underline{1111}\,\underline{1101}$

$\underline{0001}\underline{1}$ $\underline{110}\underline{1111}$

1000 0111 1101 1111 1101 0001 1110 1111

0x87DFD1EF

2) a. li x5, 0xFFFFFFFF

The value 0xFFFFFFFF represents -1 in signed 32 bit representation. Since RISC-V uses 2's complement for negative numbers, the assembler can use '-1' as immediate value instead of the original unsigned representation.
The addi instruction with 'x0' allows to effectively load '-1' into 'x5' by performing the operation 0+(-1) as this is a more valid way to represent the value to be stored.
So, the assembler translates this into 'addi x5,x0,-1'. to effectively load signed value into the register.

b. li x5,132.

The value 132 fits within the immediate value range of addi and thus can use this value directly.
The addi instruction adds the immediate value 132 to the x0. This effectively loads the value '132' to register x5 as '0+132', this is a more efficient way to ~~store~~ load values.
So, it translates to 'addi x5,x0,132'.

2) (c) li x5, 2134

As 2134 is outside the range of -2048 to 2047, it is actually translating this immediate value loading into storing the upper part (20 bits) of the immediate value first into x5, and then adding the remaining lower part (32 bit signed), ultimately resulting to the desired value of 2134.
So, it translates into lui x5,0x1 and then addiw x5,x5,-1962.

(d) li x5, 0x2345abcd

The immediate value to be loaded into x5 is very large, and larger than the immediate value allowable range [-2048 to 2047]. So, it is translating into multiple instructions.
The translation involves:
lui x5, 0x2345b → loading the upper 20 bits into x5 which is 0x2345b000
addiw x5,x5, -1075
this instruction then adds 0xabcd to x5, typically ~~adding~~ loading the lower 12 bits. 0xabcd in signed decimal form is -1075.

3.a. 0x0019F233

0 0 1 9 F 2 3 3

0000 0000 0001 1001 1111 0010 0011 0011

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

The last 7 bits [0110011] represents R format instruction and the funct3 [14:12] is $(111)_2 = (7)_{10}$ represents that this is AND instruction.

$rs2 = (00001)_2 = (1)_{10}$

$rs1 = (10011)_2 = (19)_{10}$.

$rd = (00100)_2 = (4)_{10}$

So, the instruction is.

~~and x19,~~ and x4, x19, x1

(b) 0x06B4D763

0 6 B 4 D 7 6 3

0000 0110 1011 0100 1101 0111 0110 0011

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

The last 7 bits [1100011] represents B format instruction and funct3 [14:12] is $(101)_2 = (5)_{10}$ represents that this is bge instruction.

$rs2 = (01011)_2 = (11)_{10}$

$rs1 = (01001)_2 = (9)_{10}$

$imm = (0000001101110)_2 = (110)_{10}$

So, the instruction is

bge x9, x11, 110

c. 0x0169CF93

0 1 6 9 C F 9 3

0000	0001	0110	1001	1100	1111	1001	0011
31 30 29 28	27 26 25 24	23 22 21 20	19 18 17 16	15 14 13 12	11 10 9 8	7 6 5 4	3 2 1 0

The last 7 bits [0010011] represents I format instruction and funct3 [14:12] is $(100)_2 = (4)_{10}$ which represents the xori instruction.

$imm = (0000\ 0001\ 0110)_2 = (22)_{10}$

$rs1 = (10011)_2 = (19)_{10}$. , $rd = (11111)_2 = (31)_{10}$.

So, the instruction is

xori x31, x19, 22