Computer Architecture
Assignment - 2

1)

(a) addi x15, x22, -4

addi in I type instruction

~~[illegible]~~

imm [11:0] rs1 funct3 rd opcode

imm [11:0] [-4]	111111111100
rs1 [x22]	10110
funct3	000
rd [x15]	01111
opcode	0010011

11111111110010110000011110010011

in hexadecimal : 0xFFcb0793

b

`xor x23, x8, x9`
`xor` is R format type instruction R instruction is written in the following form

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

1. funct7: 0000000
2. rs2 [x9]: 01001
3. rs1 [x8]: 01000
4. funct3: 100
5. rd [x23]: 10111
6. opcode: 0110011

Overall: 00000000100101000100101110110011

which in hexadecimal is 0x00944bb3

c

`beq x7, x1, 240`

`beq` is B format type instruction
B instruction is written in the following form

imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode
7	5	5	3	5	7

240 in binary is 0000011110000

In B type the imm[0] is discarded (not used)

1. imm[12|10:5]: 0000111
2. rs2 [x1]: 00001
3. rs1 [x7]: 00111
4. funct3: 000
5. imm[4:1|11]: 10000
6. opcode: 1100011

Overall: 00001110000100111000100001100011
which in hexadecimal is 0x0e138863

d

`sd x6, 24(x9)`

`sd` is S format type instruction
S instruction is written in the following form

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

24 in binary is 000000011000

1. imm[11:5]: 0000000
2. rs2 [x9]: 00110
3. rs1 [x6]: 01001
4. funct3: 011
5. imm[4:0]: 11000
6. opcode: 0100011

Overall: 00000000011001001011110000100011 which in hexadecimal is 0x0064bc23

e

`jal x8, -10180`

`jal` is J format type instruction
J instruction is written in the following form

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

imm in binary is 111111101100000111100

1. imm[20|10:1|11|19:12]: 10000011110111111101
2. rd [x8]: 01000
3. opcode: 1101111

Overall: 10000011110111111101010001101111
which in hexadecimal is 0x83dfd46f

Question-2

`li rd, immediate`

`x5`≡ `t0`

a

```
li t0, -1
```

The equivalent disassembled code is

```
addi x5 x0 -1
```

This occurs because we want to load `-1` to `t0`. `t0` is ABI name for `x5`. We want to load -1 to `x5`. This can be converted to `addi` operation. Since `x0` always contains 0 we can translate the given instruction to `addi x5 x0 -1`

b

```
li t0, 0xFFFFFFFF
```

The equivalent disassembled code is

```
addi x5 x0 -1
```

This is because `0xFFFFFFFF` is a 32 bit integer for `-1`. Since, `li` takes either 32 bit or 16 bit signed integers, we are given 32 bit integer which represents `-1`. We want to load -1 to `x5`. This can be converted to `addi` operation. Since `x0` always contains 0 we can translate the given instruction to `addi x5 x0 -1`

c

```
li t0, 223
```

The equivalent disassembled code is

```
addi x5 x0 223
```

This is because 223 lies in the range -2^{11} to $2^{11}-1$. It can be directly written as `addi x5 x0 223` since `x0` always gives `0`

d

```
li    t0, 1234
```

The equivalent disassembled code is

```
addi x5 x0 1234
```

This is because 1234 lies in the range -2^{11} to $2^{11}-1$. It can be directly written as `addi x5 x0 223` since `x0` always gives `0`

e

```
li t0, 0x123456000
```

The equivalent disassembled code is

```
lui x5 0x92
addiw x5 x5 -1493
slli x5 x5 13
```

Let us understand is line by line.
1. `lui x5 0x93`
This loads 0x93000 in x5. `lui` appends 12 bits (0s) to the end. The Value in x5 after this operation is 598016
2. `addiw x5 x5 -1493`
addiw takes 12 bit immediate and adds it to 32 bit `rs1` (x5) and writes it into rd (x5). The Value in x5 after this operation is 596523
3. `slli x5 x5 13`
This left shits the value in `x5` by 13. `x5 = x5 << 13`. The Value in x5 after this operation is 4886716416 which is `0x0000000123456000` in hex.

Problem-3

1. `lui x1, 0x10000`
   Loads the given memory address in `x1`
2. `lhu x3, 0(x1)`
   `lhu` loads the unsigned halfword (16 bits) into `x3`. Since the memory address at x1 is `0x39933939a55aa5a5`. We load the first 16 bits, which are `a5a5`. Thus, the value at `x3` after this operation is `0x000000000000a5a5`
3. `lh x3, 0(x1)`
   `lh` loads the signed halfword (16 bits) into `x3`. Since the memory at x1 is `0x39933939a55aa5a5`. We load the first 16 bits, which are `a5a5`. Since `a5a5` in binary is `1010010110100101`. after the sign extension we get `0xffffffffffffa5a5` because `a5a5` has a negative sign.
4. `lh x3, 2(x1)`
   `lh` loads the signed halfword (16 bits) into `x3`. Since the memory at x1 (16 bits only) after offset of 2 bytes is `0xa55a` which also has a negative sign. After sign extension, we get `0xffffffffffffa55a` in `x3`
5. `ld x3, 0(x1)`
   `ld` loads the doubleword from the memory address at x1. So, we get `0x39933939a55aa5a5` into `x3`.
6. `ld x3, 4(x1)` `ld` loads the doubleword from the memory address at x1 with offset 4 bytes, that is from `0x10000004`. So, we get `0x3993393939933939` into `x3`.
7. `lbu x3, 7(x1)`
   `lbu` loads 1 byte (unsigned) from memory address at x1 after offset of 7 bytes, that is `0x10000007`. So, we get `0x0000000000000039` into `x3`.
8. `lb x3, 7(x1)` `lb` loads 1 byte (signed) from memory address at x1 after offset of 7 bytes, that is `0x10000007`. Sine the value is `0x39` the sign is positive (0 at most significant bit), after the sign extension we get `0x0000000000000039` into `x3`.

9. `lb x3, 6(x1)` `lb` loads 1 byte (signed) from memory address at x1 after offset of 7 bytes, that is `0x10000007`. Sine the value is `0x93` the sign is negative (1 at most significant bit), after the sign extension we get `0xffffffffffffff93` into `x3`.