Answer 1)

Assumptions –

```
A[0] = x27

B[0] = x30

C[0] = x31

f = x5

g = x6

h = x7

I = x28

j = x29
```

In order to fetch the address, we multiply the position with 4 to generate the offset address w.r.t the address of Array[0]

Temporary registers for storing the temp values are x3 and x4

The machine is a 32bit machine hence, we are considering sw and lw instead of ld and sd.

a)

4 address spaces (pointers) allocated per integer makes A[10] = 4*10 spaces

= 40 bytes as offset

Read memory and update register

lw x5, 40(x27)

b)

Similarly, A[17] = 4 * 17 = 68 bytes as offset

Writing register value to memory (Store)

sw x5, 68(x27)

c)

add x7, x5, x6

d) C[g] = A[i+j+31] (given)

A [31] = 4 * 31 = 124 bytes as offset

add x3, x28, x29	- adding I and J
addi x3, x3, 31	- incrementing pointer by 31 places
slli x3, x3, 2	- fetch offset form by shift left of 2 places
add x3, x27, x3	- add the base address of A [0]
lw x3, 0(x3)	- load x3 register with value of x3
slli x4, x6, 2	- fetch value (g) in offset by shift left by 2 places
add x4, x31, x4	- add the base address of C [0]
sw x3, 0(x4)	- store x4 with the value stored in register x3

e)

i) `f = g - A[B[9]]`

B[9] = 9* 4 = 36 offset

lw x3, 36(x30)	- load x3 with address of B9
slli x3, x3, 2	- Fetch the value in offset by shifting left by 2
add x3, x3, x27	- Add base value of A[0] to this offset value
lw x3, 0(x3)	- load register x3 with value at this position
sub x5, x6, x3	- fetching f by subtracting value of x3 from g

(ii) `f = g - A[C[8] + B[4]]`

C[8] = 8 * 4 = 32 offset and B[4] = 4 * 4 = 16 offset

lw x3, 32(x31)	- load x3 with address of C8
lw x4, 16(x30)	- load x4 with address of B4
add x3, x3, x4	- add these values in x3

Repeat the last 4 steps from i)

slli x3, x3, 2	- Fetch the value in offset by shifting left by 2
add x3, x3, x27	- Add base value of A[0] to this offset value
lw x3, 0(x3)	- load register x3 with value at this position

sub x5, x6, x3 - fetching f by subtracting value of x3 from g

(iii) `A[i] = B[2i+1], C[i] = B[2i]`

B[2i] = 2 * 4 * i = 8i offset and C[i] = i * 4 = 4i offset (where i is x28)

```
slli x3, x28, 3        - Fetch the value 8i in offset by shifting left by 3
add x3, x3, x30        - Add base value of B [0] to this offset value
slli x4, x28, 2        - Fetch the value 4i in offset by shifting left by 2
add x4, x4, x31        - Add base value of C [0] to this offset value
sw x3, 0(x4)           - load register x4 with value at register x3
```

Similarly, follow the above steps for A instead of C

```
slli x3, x28, 3        - Fetch the value 8i in offset by shifting left by 3
add x3, x3, 4          - Adding 4 from B[2i+1] offset bytes
add x3, x3, x30        - Add base value of B [0] to this offset value
slli x4, x28, 2        - Fetch the value 4i in offset by shifting left by 2
add x4, x4, x27        - Add base value of A [0] to this offset value
sw x3, 0(x4)           - load register x4 with value at register x3
```

(iv) `A[i] = 4B[i-1] + 4C[i+1]`

4B[i] = 4 * 4 * i = 16i offset and 4C[i] = 4 * 4 * i = 16i offset

```
addi x3, x28, -1       - decrementing pointer by 1 place
slli x3, x3, 4         - Fetch the value 16i in offset by shifting left by 4
add x3, x3, x30        - Add base value of B [0] to this offset value

addi x4, x28, 1        - incrementing pointer by 1 place
slli x4, x4, 4         - Fetch the value 16i in offset by shifting left by 4
add x4, x4, x31        - Add base value of C [0] to this offset value
slli x4, x28, 2        - Fetch the value 4i in offset by shifting left by 2
```

add x3, x3, x4 - adding B and C related value to register x3

lw x3, 0(x3) - load register x3 with value at register x3

add x4, x4, x27 - Add base value of A [0] to this offset value

sw x3, 0(x4) - load register x4 with value at register x3

(v) `f = g - A[C[4] + B[12]]`

C[4] = 4 * 4 = 16 offset and B[12] = 12 * 4 = 48 offset

lw x3, 16(x31) - load x3 with address of C4

lw x4, 48(x30) - load x4 with address of B12

add x3, x3, x4 - add these values in x3

slli x3, x3, 2 - Fetch the value in offset by shifting left by 2

add x3, x3, x27 - Add base value of A[0] to this offset value

lw x3, 0(x3) - load register x3 with value at this position

sub x5, x6, x3 - fetching f by subtracting value of x3 from g

Answer 2)

Given

`x5 = 0x00000000AAAAAAAA = 10101010101010101010101010101010`

`x6 = 0x1234567812345678 = 00010010001101000101011001111000`
`00010010001101000101011001111000`

a)

`srli x7, x5, 16`

shift right with 0 = $(1010101010101010)_2 = (000000000000AAAA)_{16}$

`addi x7, x7, -128`

$(128)10 = (10000000)_2$

$(-128)10 = (11111111111111111111111101111111)_2 + (1)_2$

$= (11111111111111111111111110000000)_2$

Addition immediate = x7 + $(-128)_2$

$= (00000000000000001010101010101010)_2$

$+ (11111111111111111111111110000000)_2$

$= (00000000000000001010101000101010)_2$

$= (000000000000AA2A)_{16}$

`srai x7, x7, 2`

shift right with sign bit = $(00000000000000000010101010001010)_2$

$= (0000000000002A8A)_{16}$

`and x7, x7, x6`

$= (00000000000000000010101010001010)_2$ AND

$(00010010001101000101011001111000)_2$

$= (00000000000000000000001000001000)_2$

$= (0000000000000208)_{16}$

b)

`slli x7, x6, 4`

Shift left with 0 =

$(00100011010001010110011110000000100100011010001010110011110000000)_2$

$= (2345678123456780)_{16}$

c)

`srli x7, x5, 3`

Shift right with 0 = $(00010101010101010101010101010101)_2 = (15555555)_{16}$

`andi x7, x7, 0xFEF`

$(FEF)_{16} = (00000000000000000000111111101111)_2$

$(FEF)_{16}$ & $(00010101010101010101010101010101)_2$

$= (00000000000000000000010101000101)_2$

$= (545)_{16}$

Answer 3)

Instruction	**func7 or immediate (imm)**	**source register (rs2)**	**source register (rs1)**	**func3**	**destination register (rd) or immediate (imm)**	**opcode (op)**
	7	**5**	**5**	**3**	**5**	**7**
`add x5, x6, x7`	0000 000	0 0110	0011 1	000	0010 1	011 0011
`addi x8, x5, 512`	0010 0000 0000		0010 1	000	0100 0	001 0011
`ld x3, 128(x27)`	0000 1000 0000		1101 1	011	0001 1	000 0011
`sd x3, 256(x28)`	0001 000	0 0011	1110 0	011	0000 0	010 0011
`beq x5, x6 ELSE`	---	0 0110	0010 1	000	1000 0	110 0011
`add x3, x0, x0`	0000 000	00000	00000	000	0001 1	011 0011
`auipc x3, FFEFA`	1111 1111 1110 1111 1010				0001 1	001 0111
`jal x3 ELSE`	0000 0001 0000 0000 0000				0001 1	110 1111

Instruction	Format	8 hex char instruction	32-bit instruction
add x5, x6, x7	R	007382B3	0000 0000 0111 0011 1000 0010 1011 0011
addi x8, x5, 512	I	20028413	0010 0000 0000 0010 1000 0100 0001 0011
ld x3, 128(x27)	I	080DB183	0000 1000 0000 1101 1011 0001 1000 0011
sd x3, 256(x28)	S	103E3023	0001 0000 0011 1110 0011 0000 0010 0011
beq x5, x6 ELSE	SB	00628863	0000 0000 0110 0010 1000 1000 0110 0011
add x3, x0, x0	R	000001B3	0000 0000 0000 0000 0000 0001 1011 0011
auipc x3, FFEFA	U	FFEFA197	1111 1111 1110 1111 1010 0001 1001 0111
jal x3 ELSE	UJ	010001EF	0000 0001 0000 0000 0000 0001 1110 1111

Answer 4)

a) Value of A = x5

Base address of C = x11

lw x5, 0(x11) - assign x5 the value stored at C [0]

slli x5, x5, 16 - shifting the bits to left by 16 places

b) x3=0 and x4 = 0XFFFFFFFFFFFFFFFF (given)

i) Load 6 bits from 12th to 7th from x3

addi x10, x0, 0x3F i.e., 0011 1111 (required for masking the bits)

slli x10, x10, 7 - shift the bits by 7 places i.e., 12th to 7th

and x11, x10, x3 - fetch only desired bits in x11

We need to shift it for (28-12) i.e., 16 places.

ii) Store 6 bits from 28th to 23rd in x4

slli x11, x11, 16 - shift by 16 places

slli x10, x10, 16 - shifting masking bits by 16 places

xori x10, x10, -1 - inverting the masked bits

(Gives 1 when both bits are same)

and x4, x4, x10 - empty the 28^{th} to 23^{rd} place in register to 0

or x4, x4, x11 - anything OR with 0 is the number itself

c) xori x5, x6, -1 //invert the bits at x5 and x6 & store them in x5

Answer 5)

PC = `0x60000000`$_{hex}$

We can start storing values from PC + 1 position i.e., $0x60000004_{hex}$

a)

The immediate value for jal instruction can be 20bits long i.e., between $(-2)^{19}$ to $(2)^{19}$ -1

Hence, the range of possible values of offset lies between $(-2)^{20}$ to $(2)^{20}$ -2

$= (100000)_{16}$ to $(FFFFC)_{16}$

Therefore, the range of PC after jump and link is

From $(60000004)_{16} – (100000)_{16}$ to $(60000000)_{16} – (FFFFC)_{16}$

Final range = $(5FF00004)_{16}$ to $(600FFFFE)_{16}$

b)

The immediate value for beq instruction can be 12bits long i.e., between $(-2)^{11}$ to $(2)^{11}$ -1

Hence, the range of possible values of offset lies between $(-2)^{12}$ to $(2)^{12}$ -2

$= (1000)_{16}$ to $(FFE)_{16}$

Therefore, the range of PC after branch if equal is

From $(60000004)_{16} – (1000)_{16}$ to $(60000000)_{16} – (FFE)_{16}$

Final range = $(5FFFF004)_{16}$ to $(60001002)_{16}$

Answer 6)

Assumptions –

```
LOOP:     beq x6, x0, DONE
          addi x6, x6, -1
          addi x5, x5, 2
          jal x0, LOOP
DONE:
x6 Register = 10 = i

x5 Register = 0 = acc
```

a)

```c
#include <stdio.h>
int main()
{
int i=10, acc=0;
LOOP:
{
if(i==0)        //beq x6, x0, DONE

goto DONE;
else
{
i=i-1;  //addi x6, x6, -1
acc=acc+2;  //addi x5, x5, 2
goto LOOP;  //jal x0, LOOP
}
}
DONE:
{return 0;}
}
```

b)

No. of instructions in loop : 4

No. of times loop would run : N + 1 `//(N=10,N>=0,N--)`

No. of time “Done” instruction run: 1

Total = (4* N + 1) + 1 = 4N + 2 RISK instructions will be executed.

c)

```c
#include <stdio.h>
int main()
{
int i=10, acc=0;
LOOP:
{
if(i<0)        //blt x6, x0, DONE
goto DONE;
else
{
i--;  //addi x6, x6, -1
acc=acc+2;  //addi x5, x5, 2
goto LOOP;  //jal x0, LOOP
}
}
DONE:
{return 0;}
}
```

Answer 7)

Assumptions –

A[0] = x5

B[0] = x6

&D[0] = x10

I = x7

j = x29

Temporary registers for storing the temp values are x3 and x4

First initialize counter registers to store iterations values (x7 (i) and x29 (j))

The comments/explanation are now written in this color beyond this point.

a)

addi x7, x0, 0 - i=0

Outer_Loop:

bge x7, x5, Outer_Loop_End // - i<a

addi x29, x0, 0 - j=0

Inner_Loop:

bge x29, x6, Inner_Loop_End - j<b

add x3, x7, x29 - i + j;

slli x4, x29, 4 - 4*j

add x4, x4, x10 - D[4*j]

sw x3, 0(x4) - D[4*j] = i + j

(i.e., storing the value of x3 directly to the address of x4 which is D[4*j])

addi x29, x29, 1 - j++

jal x0, Inner_Loop – Jump and return to Inner Loop

Inner_Loop_End:

addi x7, x7, 1 - i++

jal x0, Outer_Loop - Jump and return to Outer Loop

Outer_Loop_End: - End the program

b)

A[0] = x5 register = 10

B[0] = x6 register = 1

&D[0] = x10 = 0

Command	Times Executed
addi x7, x0, 0	1
Outer_Loop:	
bge x7, x5, Outer_Loop_End	11 (I (0 to 9) – 10 times + I = 10th value) Outer loop
addi x29, x0, 0	10 (1 time for each i)
Inner_Loop:	
bge x29, x6, Inner_Loop_End	20 [j = 1 (10 times) and j = 0 (10)]
add x3, x7, x29	10 (10 (I times) * 1 (j times))
slli x4, x29, 4	10 (10 (I times) * 1 (j times))
add x4, x4, x10	10 (10 (I times) * 1 (j times))
sw x3, 0(x4)	10 (10 (I times) * 1 (j times))
addi x29, x29, 1	10 (10 (I times) * 1 (j times))
jal x0, Inner_Loop	10 (10 (I times) * 1 (j times))
Inner_Loop_End:	
addi x7, x7, 1	10 (No of iterations of I)
jal x0, Outer_Loop	10 (No of iterations of I)
Outer_Loop_End:	

Total = 1+11+10+20+(6*10) +(2*10) = 122

Answer 8)

Considering the base address 0x10000000. It is 8 bits machine and could store 0×1122334455667788 as

a) Big Endian (MSB is stored last)

Digit	Address
11	0×10000000
22	0×10000001
33	0×10000002
44	0×10000003
55	0×10000004
66	0×10000005
77	0×10000006
88	**0×10000007**

b) Little Endian (MSB is stored first)

Digit	Address
88	0×10000000
77	0×10000001
66	0×10000002
55	0×10000003
44	0×10000004
33	0×10000005
22	0×10000006
11	**0×10000007**

Answer 9)

$(1234567812345678)_{16}$ = $(00010010001101000101011001111000000010010001101000101011001111000)_2$ i.e., 64 digits. Hence, we cannot do using a single register x10. The temp register considered here will be x3.

Code –

lui x3, 0x12345 //load 32 bits using this //20 bits (5digits from hex) in x3 temp register

```
addi x10, 0x678, x3    //r2 being value, add immediate value of 678 to x10 register

slli x10, x10, 32      //shift left by 32 places as the whole word is 64 digits/bits

addi x3, 0x678, x3     //r2 being value, add immediate value of 678 to x3 temp register

                       //x10 is needed to store the final value so, not touched

add x10, x3, x10       //finally add 1234567800000000 to 12345678 in hex
```

Answer 10)

Given:

$x5 = (128)_{10}$

a)

For overflow, the range should exceed $(FFFFFFFF)_{16}$ for overflow to occur

$(00000080)_{16} = (128)_{10}$

Hence,

Minimum value of x6 should = $(FFFFFFFF)_{16} - (00000080)_{16} + 1 = (FFFFFF86)_{16}$ for an overflow to occur.

b)

x5 – x6 should not be greater than 0. Hence, $x6 > (128)_{10}$ for an overflow to occur.

c)

x6 – x5 should not be greater than 0. Hence, $x6 < (128)_{10}$ for an overflow to occur.