Chapter - 1

Ques 1 :-

Given,

(c)

$C/W_x = 20$	$C/W_y = 15$
$D_x = 19\,cm$	$D_y = 20\,cm$
$R_x = \frac{19}{2}\,cm$	$R_y = \frac{20}{2}\,cm$
$D/W_x = 89$	$D/W_y = 100$
$Def/A_x = 0.023$ defects/cm^2	$Def/A_y = 0.031$ defects/cm^2

$$Yield_x = \frac{1}{\left(1+\left(\text{Defects per area} \times \frac{\text{Die Area}}{2}\right)\right)^2}$$

$$= \frac{1}{\left(1+\left(0.023 \times \frac{\text{Die Area}}{2}\right)\right)^2}$$

$$= \frac{1}{\left(1+\left(0.023 \times \frac{3.1857}{2}\right)\right)^2}$$

$$\cong 0.93$$

$$D/W = \frac{\text{Wafer Area}}{\text{Die Area}}$$

$$\text{Die Area} = \frac{\text{Wafer Area}}{D/W}$$

$$= \frac{\pi \times \left(\frac{19}{2}\right)^2}{89}$$

$$= 3.1857$$

$$Yield_y = \frac{1}{\left(1+\left(\text{Defects per area} \times \frac{\text{Die Area}}{2}\right)\right)^2}$$

$$= \frac{1}{\left(1+\left(0.031 \times \frac{\text{Die Area}}{2}\right)\right)^2}$$

$$= \frac{1}{\left(1+\left(0.031 \times \frac{3.1416}{2}\right)\right)^2}$$

$$\cong 0.91$$

$$D/W = \frac{\text{Wafer Area}}{\text{Die Area}}$$

$$\text{Die Area} = \frac{\text{Wafer Area}}{D/W}$$

$$= \frac{\pi \times \left(\frac{20}{2}\right)^2}{100}$$

$$= 3.1416$$

d) $$C/D_x = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$= \frac{20}{89 \times 0.93}$$

$$= 0.24$$

$$C/D_y = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$= \frac{15}{100 \times 0.91}$$

$$= 0.16$$

Ques 2:

	A	B	C	D
CPI_{PS}	7	2	3	6
CPI_{XB}	5	4	2	1
IC	$1\times10^6\times.3 = 3\times10^5$	$1\times10^6\times.5 = 5\times10^5$	$1\times10^6\times.1 = 1\times10^5$	$1\times10^6\times.1 = 1\times10^5$

$Rate_{PS} = 2.7\,GHz = 2.7\times10^9\,Hz$

$Rate_{XB} = 3\,GHz = 3\times10^9\,Hz$

$Instruction\ Count = 1\times10^6$

a) For Play Station,

$$Clock\ Cycles = (7\times3\times10^5 + 2\times5\times10^5 + 3\times1\times10^5 + 6\times1\times10^5$$

$$= 4\times10^6$$

$$Avg.\ CPI = \frac{4\times10^6}{1\times10^6}$$

$$= 4$$

For XBOX,

$$Clock\ Cycles = (5\times3\times10^5 + 4\times5\times10^5 + 2\times1\times10^5 + 1\times1\times10^5$$

$$= 3.80\times10^6$$

$$Avg.\ CPI = \frac{3.80\times10^6}{1\times10^6}$$

$$= 3.8$$

So, Difference $= (4 - 3.8)$

$= 0.2$

b) $$Ex\ time_{PS} = \frac{Instruction\ Count \times CPI}{Rate}$$

$$= \frac{10^6\times4}{2.7\times10^9}$$

$$= 0.00148\ s$$

$$Ex\ time_{PS} = \frac{Instruction\ Count \times CPI}{Rate}$$

$$= \frac{10^6\times3.8}{3\times10^9}$$

$$= 0.00126\ s$$

$$Difference = (0.00148 - 0.00126)$$

$= 0.22$

Execution time of a Reference processor

Reference time is provided by SPEC

Measured time is the Execution time of a given

c) $$Spec\ Ratio = \frac{Reference\ Time}{Measured\ Time}$$

$$= \frac{120}{Inst.\ Count \times CPI \times C.eT}$$

$$= \frac{120\times10^{-3}}{10^6\times4\times\frac{1}{2.7\times10^9}}$$

$= 81$

Ques-3

Processor	CPI	Rate
P1	1.5	3 GHz
P2	1.0	2.5 GHz
P3	2.2	4 GHz

Instruction Count = X

[same for all Processors]

a) $\text{Performance (IPS)} = \frac{\text{Clock Rate}}{\text{CPI}}$

$P_{P1} = \frac{3 \times 10^9}{1.5} = 2 \times 10^9$

$P_{P2} = \frac{2.5 \times 10^9}{1} = 2.5 \times 10^9$ ⇒ highest perf.

$P_{P3} = \frac{4 \times 10^9}{2.2} \cong 1.82 \times 10^9$

Rate = cycles completed per second

for P1,

3×10^9 cycles --- 1 s

∴ 1 " ... $\frac{1}{3 \times 10^9}$ s

∴ 1.5 " ... $\frac{1.5}{3 \times 10^9}$ s

[1 ins = 1.5 cycle]

So,

1 ins. takes $\frac{1.5}{3 \times 10^9}$ s to complete

$\frac{1.5}{3 \times 10^9}$ s to complete – 1 ins

∴ 1 s " " $\frac{3 \times 10^9}{1.5}$ ins (Rate / CPI)

$\text{Performance (IPS)} = \frac{\text{Clock Rate}}{\text{CPI}}$

b) $CPU_{Time} = 10s$

$CPU_{Time} = \frac{IC \times CPI}{Rate}$ (CPI: cycle)

C(P1) = $6 \times 3 \times 10^9$

C(P2) = $6 \times 2.5 \times 10^9$

C(P3) = $6 \times 4 \times 10^9$

$\Rightarrow IC_1 = \frac{\text{Cycle Count (P1)}}{\text{CPI}}$

$= \frac{6 \times 3 \times 10^9}{1.5}$

$= 1.2 \times 10^{10}$

$\Rightarrow IC_2 = \frac{\text{Cycle Count (P2)}}{\text{CPI}}$

$= \frac{6 \times 2.5 \times 10^9}{1.0}$

$= 1.5 \times 10^{10}$

$\Rightarrow IC_3 = \frac{\text{Cycle Count (P3)}}{\text{CPI}}$

$= \frac{6 \times 4 \times 10^9}{2.2}$

$= 1.09 \times 10^{10}$

c)

Processor	CPI	New CPI
P1	1.5	1.8
P2	1.0	1.2
P3	2.2	2.64

$$CPU\ Time_{new} = 10 - (10 \times .3)$$
$$= 7s$$

$$CPU_{Time} = \frac{I.C \times CPI}{Freq.}$$

$$\Rightarrow Freq. = \frac{IC \times CPI}{CPU_{Time}}$$

$$F(P_1) = \frac{IC \times CPI}{CPU_{Time}} = \frac{1.2 \times 10^{10} \times 1.8}{7} \simeq 3.09\ GHz$$

$$F(P_2) = \frac{IC \times CPI}{CPU_{Time}} = \frac{1.5 \times 10^{10} \times 1.2}{7} \simeq 2.57\ GHz$$

$$F(P_3) = \frac{IC \times CPI}{CPU_{Time}} = \frac{1.09 \times 10^{10} \times 2.64}{7} \simeq 4.11\ GHz$$

Quen-4 :

$$CPU_{Time} = 250\ s$$

Add	Sub	left-Shift
70 s	85 s	40 s

$= 105s$

mine. $\Rightarrow (250 - 105) = 55$

a) $T_{new} = (70 \times 0.8) + 85 + 40 + 55$

$= 236s$

time reduced $= (250 - 236) = 14\ s$

$= \frac{14}{250} \times 100\% \simeq 5.47\%$

b) $T_{new} = (250 \times 0.8) = 200\ s$

$T_{add} + T_{sub} + T_{mine} = 70 + 85 + 55 = 210s > T_{new}.$

NO

Ques 5:

$$MIPS = \frac{Clock\ Rate}{CPI \times 10^6}$$

Rate (P1) = 4 GHz = 4×10^9 Hz

CPI (P1) = 0.9

Ins. Count = 5×10^9

$$MIPS(P1) = \frac{4 \times 10^9}{0.9 \times 10^6}$$

$$= 4.44 \times 10^3$$

Rate (P2) = 3 GHz = 3×10^9 Hz

CPI (P2) = 0.75

Ins. Count = 1×10^9

$$MIPS(P2) = \frac{3 \times 10^9}{0.75 \times 10^6}$$

$$= 4 \times 10^3$$

$$P(P1) = \frac{4 \times 10^9}{5 \times 10^9 \times 0.9} = 0.889$$

$$P(P2) = \frac{3 \times 10^9}{1 \times 10^9 \times 0.75} = 0.44$$

MIPS(P1) > MIPS(P2)

Per. (P1) > Per. (P2)

True

Ques -6

$$power = CL \times V^2 \times F$$

$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V \times 0.81)^2 \times F \times 0.81}{C_{old} \times V^2 \times F}$$

$$= \frac{C_{old} \times 0.85 \times V^2 \times 0.81^2 \times F \times 0.81}{C_{old} \times V^2 \times F}$$

$$= 0.4605$$

about

The new processor uses almost Half (0.4605) power of the old processor.

Ques - 7

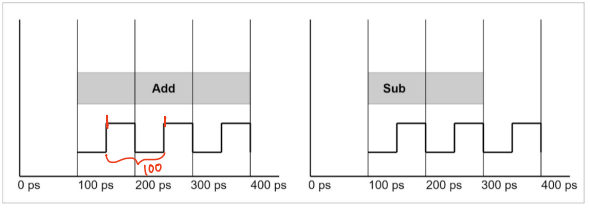


Figure 1: Represents 1 add instruction

Figure 2: Represents 1 sub instruction

Figure 1: Represents 1 add instruction
Figure 2: Represents 1 sub instruction

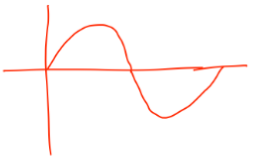
Program **A** is divided into two classes according to their **CPI** (Add and Sub). The **instruction counts** are 21 and 3 respectively. Reference for **program A** is 1080ps.
Now, answer the following questions,

a) What is the Clock period? **Hint**: follow any of the figures
b) What is the frequency?
[1]
c) What is the CPI for Add and Sub ?
[2]
d) What is the Avg. CPI?
[2]
e) Find out the execution time of the program?
[2]
f) Find the SPEC ratio?
[1]
g) If you want to improve the performance by 1.2 times, what improvement do you need to include in the program's add operation?
[2]

Program A

Class	Add	Sub
CPI	3	2
IC	21	3

$\frac{(21\times3)+(3\times2)}{24} = 2.875$



Duration between two rising edges = 100 ps = 100×10^{-12} s

Freq = $\frac{1}{100\times10^{-12}} = 10^{10}$ Hz $= 10\times10^{9}$ Hz $= 10$ GHz

Clock cycle per ins. → 3, 2

2.875

Clock Cycle

$\Rightarrow$ Ex/CPU Time = Ins./Prog × CPI × Clock Period

$= 24 \times 2.875 \times 100\times10^{-12}$

$= 6.9\times10^{-9}$ s / 6900 ps

Spec Ratio = $\frac{\text{Ref. time}}{\text{CPU time / Execution time}}$

g)

$\text{Time}_{\text{improved}} = \frac{T_{\text{affected}}}{n} + T_{\text{unaff.}}$

$\Rightarrow \frac{6.9\times10^{-9}}{1.2} = \frac{6.0375\times10^{-9}}{n} + 8.625\times10^{-10}$

$\Rightarrow n = 1.235$

$= \frac{1080 \text{ ps}}{6900 \text{ ps}}$

$= 0.1565$

$T_{\text{aff.}} = 21 \times 2.875 \times 100\times10^{-12}$

$= 6.0375\times10^{-9}$

Chapter-2

Question - 1:

Construct the equivalent RISC-V code of the following C code. Once you have the RISC-V code, identify type of each instruction and encode them accordingly.

```
A[7] = A[2] + A[B[8]] + 10;
B[i] = A[3] - 8;
```

Base addresses of array A and B are in register X_{20} and X_{21} and i is in register X_{22}

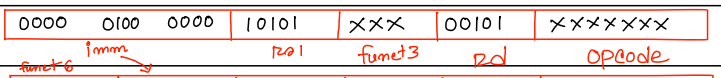
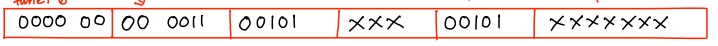
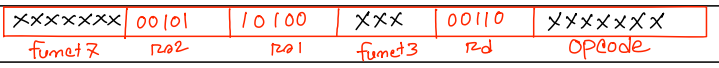
A = X_{20}
B = X_{21}
i = X_{22}

Line-1

```
ld   X5, 64(X21)     // B[8]
Slli X5, X5, 3       // offset calculation (X5 * 2^3)
Add  X6, X20, X5     // physical loc. (Base + offset)
ld   X6, 0(X6)       // A[B[8]]

ld   X5, 16(X20)     // A[2]

Add  X5, X5, X6      // A[B[8]] + A[2]

Addi X5, X5, 10      // A[B[8]] + A[2] + 10

Sd   X5, 56(X20)     // A[7] = A[B[8]] + A[2] + 10
```

Line-2

```
ld   X6, 24(X20)     // A[3]
Addi X6, X6, -8      // A[3] - 8

Slli X7, X22, 3      // offset calc => X22 * 2^3
Add  X7, X7, X21     // Physical Add. calc

Sd   X6, 0(X7)       // B[i] = A[3] - 8
```

No.	Instruction	Type	Encoding
1	ld x_5, 64(x_{21})	I ✓	
2	Slli x_5, x_5, 3	I ✓	
3	Add x_6, x_{20}, x_5	R ✓	
4	ld x_6, 0(x_6)	I	
5	ld x_5, 16(x_{20})	I	
6	Add x_5, x_5, x_6	R	
7	Addi x_5, x_5, 10	I	
8	Sd x_5, 56(x_{20})	S ✓	
9	ld x_6, 18(x_{20})	I	
10	Addi x_6, x_6, -8	I ✓	
11	Slli x_7, x_{22}, 3	I	
12	Add x_7, x_7, 21	R	
13	Sd x_6, 0(x_7)	S	

8 = 1000

+8 = 0 1000

+8 = 0000 0000 1000

1111 1111 0111

+1

-8 = 1111 1111 1000

Encode rest of the instructions yourself.

Question - 2:

Construct the equivalent RISC-V code of the following C code.

```
for (i = 8; i > 0 ; i--) {
  if ( A[i] == i){
    A[2] = A [B[3]] ;
  }
}
```

Base addresses of array A and B are in register X_{20} and X_{21} . Also consider i is in register X_{22}.

A = X_{20}
B = X_{21}
i = X_{22}

```
  Addi  X22, X0, 8
Loop:
   Beq  X22, X0, Exit

  If:
  Slli  X5, X22, 3      // offset calc.
  Add   X5, X5, X20     // Base + offset
  ld    X6, 0(X5)       // X6 = A[i]
  Bne   X6, X22, inc

  ld    X7, 24(X21)     // X7 = B[3]
 Slli   X7, X7, 3       // X7 = X7 x 2^3
 Add    X7, X7, X20     // Base + offset
 ld     X5, 0(X7)       // X5 = A[B[3]]
 sd     X5, 16(X20)     // A[2] = X5

inc:
Addi X22, X22, -1
Beq  X0, X0, Loop

Exit :
```

Question - 3:

Construct the equivalent RISC-V code of the following C code.

```
if ( A[i] < i){
    A[2] = A [B[3]] ;
}
```

Base addresses of array A and B are in register X_{20} and X_{21} . Also consider i is in register X_{22}.

```
If :
slli  x5, x22, 3      // offset calc.
Add   x5, x5, x20     // Base + offset
ld    x6, 0(x5)       // x6 = A[i]
BGE   x6, x22, Exit

ld    x7, 24(x21)     // x7 = B[3]
slli  x7, x7, 3       // x7 = x7 x 2^3
Add   x7, x7, x20     // Base + offset
ld    x5, 0(x7)       // x5 = A[B[3]]
sd    x5, 16(x20)     // A[2] = x5

Exit :
```

A = X_{20}
B = X_{21}
i = X_{22}

Question - 4:

Construct the equivalent RISC-V code of the following C code.

```
if ( A[3] != A[6]){
    if (A[3] == 0) {
        A[3] = A[3] + 2;
    }else{
        A[6] = A[6] / 16;
    }
}else{
    A[6] = A[6] * 8
}
```

Base addresses of array A and B are in register X_{20}

If 1:

```
ld   x28, 24(x20)      // x28 = A[3]
ld   x29, 48(x20)      // x29 = A[6]
Beq  x28, x29, Else1
```

If 2:

```
ld   x5,  24(x20)      // x5 = A[3]
BNE  x5,  x0,  Else2
```

→ A[3] already loaded into x_5 and x_5 is unchanged till now.

```
Addi x5, x5, 2
Sd   x5, 24(x20)       // A[3] = x5 + 2
Beq  x0, x0, Exit
```

Else 2:

```
ld   x5, 48(x20)       // x5 = A[6]
SRLI x5, x5, 4         // x5 = x5 / 2^4
Sd   x5, 48(x20)       // A[6] = x5
Beq  x0, x0, Exit
```

Else 1:

```
ld   x28, 48(x20)      // x28 = A[6]
Slli x28, x28, 3       // x28 = x28 * 8
Sd   x28, 48(x20)      // A[6] = x28
```

Exit:

Question - 5:

Translate the following code written in C programming language into instructions sequence written in RISC-V Assembly. The values in the variables a, b, and c inside the add function are stored in the argument registers $\$X_{10}$, $\$X_{15}$, and $\$X_{12}$ respectively. Also, the values in the variables x and y inside the are stored in the argument registers $\$X_{13}$ and $\$X_{14}$ respectively. The return value is stored in the return register $\$X_{11}$ for both functions.

int max(int x, int y) {
  if(x > y)
  return x;
  else
  return y;
}
	int calc(int a, int b, int c) {
	  return a + max(b,c)
	}

```
Calc:
    Addi   Sp, Sp, -8
    sd     X1, 0(SP)
    Addi   X13, X15, 0      [X13 = X15 + 0]   (b)
    Addi   X14, X12, 0      [X14 = X12 + 0]   (c)
    Jal    X1, max
    Add    X11, X10, X11
                 a   return from max
    ld     X1, 0(SP)
    Addi   Sp, Sp, 8
    Jalr   X0, 0(X1)
```

```
Max:
    Blt    X13, X14, maxElse      (x, y)
    Addi   X11, X13, 0
    Beq    X0, X0, ExitMax

maxElse:
    Addi   X11, X14, 0

ExitMax:
    Jalr   X0, 0(X1)
```

Question - 6:

Construct the equivalent RISC-V code of the following C code.

```
Main () {
    int x = 0;          X20
    int y = 9;          X21
    int z = addition(x, y);   X13, X14
}                       X22

int addition (int a, int b) {
    int c = a + b;      X21
    return c;
}
```

Variables x, y, z are stored in X_{20}, X_{21} and X_{22} registers. Argument x, y are passed using register X_{13}, X_{14}

Variable c from the addition function also uses register X_{21}

```
Addi  X20, X0, 0      // X20 = 0
Addi  X21, X0, 9      // X21 = 9

Addi  X13, X20, 0     // X13 = x
Addi  X14, X21, 0     // X14 = y
Jal   X1, addition
Addi  X22, X10, 0     // Z = X10
```

```
addition:
Addi  SP, SP, -8
ld    X21, 0(SP)      // Storing the value of saved reg.
Add   X21, X13, X14   // a = a + b
Addi  X10, X21, 0     // storing the return value.
sd    X21, 0(SP)      // restoring prev. value
Addi  SP, SP, 8
Jalr  X0, 0(X1)       // returning control to the caller.
```

Ques-7

Write RISC-V assembly code that checks if the number stored in register $\mathbf{X_{25}}$ is **even** or not. If **even** then store **1** in register $\mathbf{X_{26}}$ otherwise store **0**.

0000 ..... 0001 = 1

```
Addi  X27, X0, 1
And   X27, X25, X27 // LSB bit required
BNE   X27, X0, else
Addi  X26, X0, 1
Beq   X0, X0, Exit

else:
Addi  X26, X0, 0
Exit:
```

You can also use Slli / Srli to figure it out.

Quer-8

ADD X_{25} , X_{25} , X_0 . Can you make this instruction faster? If yes, Write the updated instruction?

Yes, ADDI x_{25}, x_{25}, 0

0 is hard wired here.

Quer-9

Memory Location	Code	Line Number	Machine Code
	ADDI $\mathbf{X_5}$, X_0 , 5	1	
# 7064	ADDI $\mathbf{X_6}$, X_0 , 1	2	
# 7068	ADDI $\mathbf{X_{25}}$, X_0 , 0	3	
# 7072	Loop: BLT $\mathbf{X_5}$, $\mathbf{X_6}$, loopBreak	4 -3×4 =-12	0 | 000 000 | 00110 | 00101 | XXX | 1000 | 0 | XXXXXXX (imm 12 | imm [10:5] | rs2 | rs1 | imm [4:1] | imm [11])
# 7076	ADDI X_{25} , X_{25} , 1	5	
#7080	ADDI X_5 , X_5 , -1	6	
	BEQ $\mathbf{X_0}$, $\mathbf{X_0}$, Loop	7	1 | 111 111 | 00000 | 00000 | XXX | 1010 | 1 | XXXXXXX (12 | 10:5 | 4:1 | 11)
	loopBreak:	8 +4×4 =16	

a) What is the value of **PC** while executing line2? Answer:

7064 [2]

b) Fill up the machine codes corresponding to line4 and line7 in the table above. [5]

16 = 0000 0001 0000

+16 = 0 0000 0001 0000 x

(12 \| 11 \| 10:5 \| 4:1)

12 = 0000 0000 1100

+12 = 0 0000 0000 1100

= 1 1111 1111 0011

+ 1

-12 = 1 111111 1010 0

(12 \| 11 \| 10:5 \| 4:1 \| 0x)

Quer -10

# Location		Line No.
	Loop :	
# 80000	Slli X_{10}, X_{22}, 3	1
# 80004	Add X_{10}, X_{10}, X_{25}	2
# 80008	Ld X_9, 0(X_{10})	3
# 80012	Bne X_9, X_{24}, Exit	4
# 80016	Addi X_{22}, X_{22}, 1	5
# 80020	Beq X_0, X_0, loop	6
	Exit:	
# 80024	⟶	7

rs1 rs2

SB type instructions are — Beq X_0, X_0, loop

Bne X_9, X_{24}, Exit

For Beq,

We are moving 5 instruction upward.

Hence, $-5 \times 4 = -20$

upward direction

20 = 0000 0001 0100

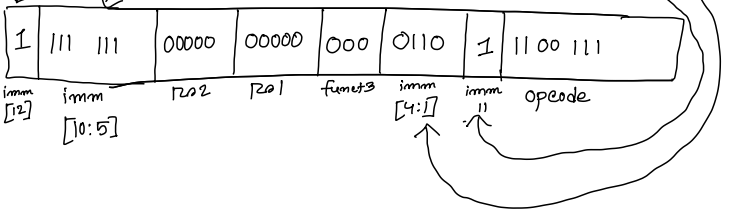
+20 = 0 0000 0001 0100

1 1111 1110 1011

+ 1

1 1111 1110 1100

12 | 11 | 10:5 | 4:1 | 0 x



For Bne,

We are moving 3 instructions downward

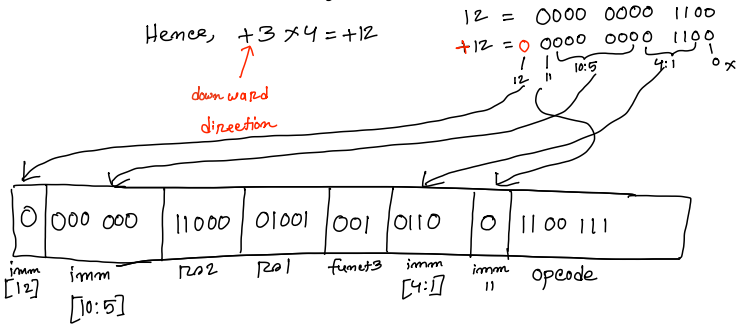
Hence, $+3 \times 4 = +12$

downward direction

12 = 0000 0000 1100

+12 = 0 0000 0000 1100

12 | 11 | 10:5 | 4:1 | 0 x



Ques-11

Construct the equivalent RISC-V code of the following C code.

```
Main () {
  addition(10, 7, 12);
}

int addition (int a, int b, int c) {
    int c = 5;
    int d = max(a, c) + b;
    int e = c + d;
    return e;
}

int max (int a, int b) {
   if(a > b)
        return a;
   else
        return b;
}
```

Arguments for **addition** are passed using register X10, X13, X14. Variables c, d, e are stored in X20, X21, X22 registers. Return value for addition should be stored in X10.

Arguments for **max** are passed using register X10, X13. Return value for **max** should be stored in X10.

```
addi   X10, X0, 10
addi   X13, X0, 7
addi   X14, X0, 12
Jal    X1, addition
```

putting the values in respective arg. registers for addition call

```
Addition:
Addi   SP, SP, -32
  sd   X1, 24(SP)
  sd   X20, 16(SP)
  sd   X21, 8(SP)
  sd   X22, 0(SP)
Addi   X20, X0, 5        // c = 5
Addi   X5, X13, 0        // b = X5 ; later we have to use b
Addi   X10, X10, 0       // redundant
Addi   X13, X20, 0       // storing the values in arg. reg for Max.
```

these three are saved registers.

```
Jal    X1,  Max
Add    X21, X10, X5      // d = X10 + b
Add    X22, X20, X21     // e = c+d
Addi   X10, X22, 0       // store the return value.

ld     X22, 0(SP)
ld     X21, 8(SP)
ld     X20, 16(SP)
ld     X1,  24(SP)
Addi   SP,  SP, 32
Jalr   X0,  0(X1)        // return control to caller.
```

```
Max:
Blt    X10, X13, maxElse   (a<b)
Addi   X10, X10, 0         // a is already in X10. Hence, redundant line.
Beq    X0,  X0,  exitMax

maxElse:
Addi   X10, X13, 0

exitMax:
Jalr   X0,  0(X1)
```

Ques-12

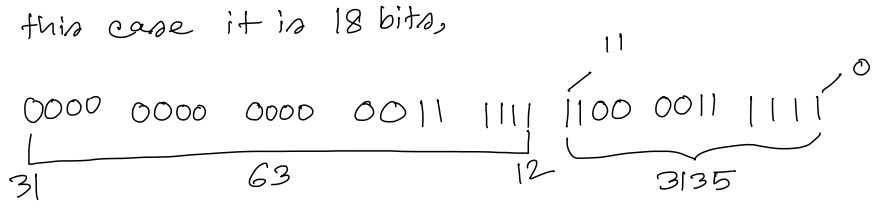
Write necessary RISC-V instructions to store the value $(1111\ 1111\ 0000\ 1111\ 11)_2$ in X20 register.

(16 bits)

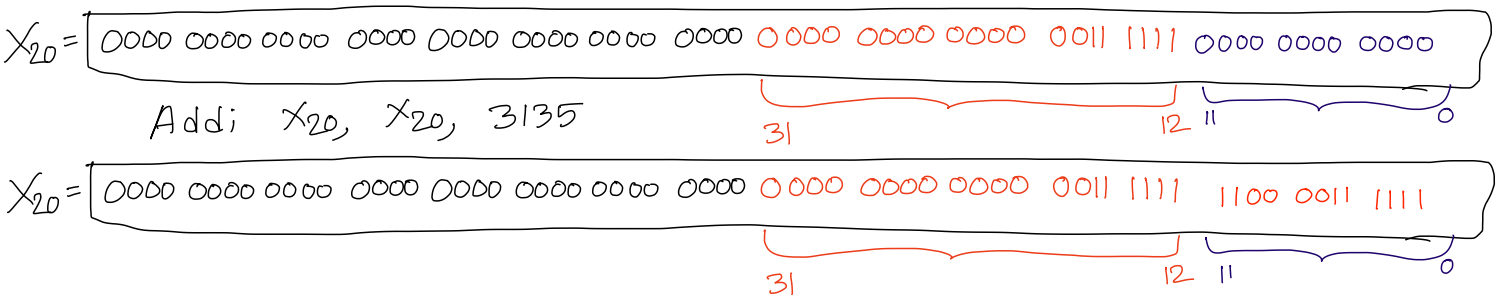
Answer2:

Check if the given number requires more than 12 bits to store.

In this case it is 18 bits,



Lui X_{20}, 63



Quen-13

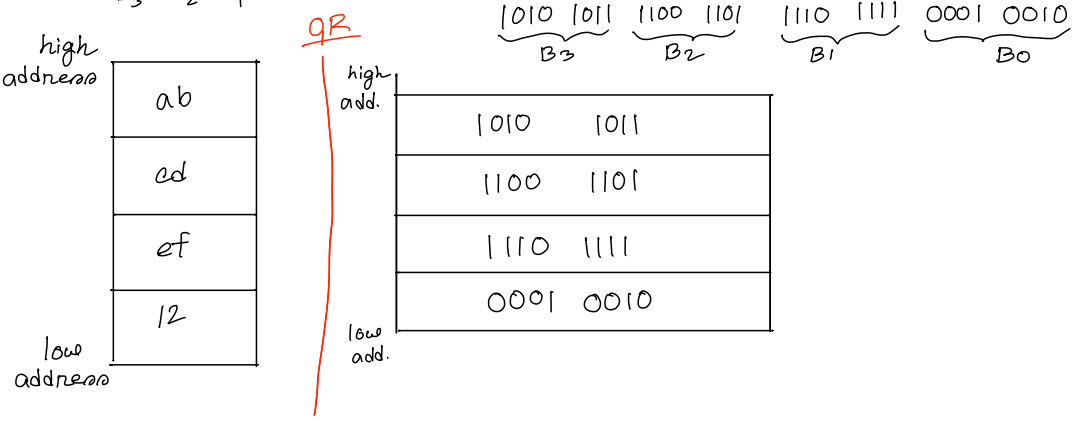
Show how the value 0xabcdef12 would be arranged in memory in RISC-V machine.

Answer3:

RISC-V follows Little endian approach to store data in memory

$\Rightarrow$ $(abcdef12)_{16}$

B_3 B_2 B_1 B_0



Ques-14

For the RISC-V assembly instructions below, what is the corresponding C/high level statement?

slli x30, x5, 3 $\Rightarrow X_{30} = f \times 8$ add x30, x10, x30 $\Rightarrow X_{30} = Base_A + 8f$ slli x31, x6, 3 $\Rightarrow X_{31} = g \times 8$ add x31, x11, x31 $\Rightarrow X_{31} = Base_B + 8g$ ld x5, 0(x30) $\Rightarrow f = A[f]$ addi x12, x30, 8 $\Rightarrow X_{12} = X_{30} + 8$ ld x30, 0(x12) $\Rightarrow X_{30} = A[f+1]$ add x30, x30, x5 $\Rightarrow X_{30} = A[f+1] + A[f]$ sd x30, 0(x31) $\Rightarrow B[g] = A[f+1] + A[f]$	Assume that the variables f, g, h, i, and j are assigned to registers x5, x6, x7, x28, and x29, respectively. Assume that the base address of the Arrays A and B are in registers x10 and x11, respectively.

$B[g] = A[f+1] + A[f]$