Answer 1)

Given –
Instruction word: 0x00c6ba23 i.e., 11000110101110100010011
where,

Hex code	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
0x00c6ba23	0010 100	0 1100	0110 1	011	0000 0	010 0011

This translates to sd x12, 20(x13)

Answer 1.1)
SD instructions have ALUOp1 = 0 and ALUOp0 = 0
30th bit = 0 and [14:12] bits = 011

Due to the presence of 00 in ALUOp0, add instruction is the output of it denoted by 0010.

Answer 1.2)
This exceeds 16 bits i.e., double word. Hence, we add 4 to the instruction memory
PC <- PC + 4
Path Taken:
PC serves as the PC+4 adder's first input. The adder's second input is 4, which. The mix PCSrc receives the output PC+4 and delivers it back to PC while disregarded all other input.
(sd does not take any branch)

Answer 1.3)
We need 3 Mux i.e., PCSrc, ALUSrc and MemToReg to write back to register file

Mux	Input1	Input2	Output
PCSrc	PC + 4	PC+ (SignExtended(Immediate)<<1) = PC+(0x0000000000000014 << 1) = PC+(0x0000000000000028) Or PC + 40	PC+4
ALUSrc	RS2 = REG[x12]	SignExtended(Immediate) = 0x0000000000000014	0x0000000000000014
MemToReg	REG[x13] + 0x0000000000000014	Unknown	Unknown

Answer 1.4)
For the sd instruction, the ALU unit is tasked to compute the memory address to which the data needs to be stored.

ALU takes inputs:
- ALU Control Unit -> 0010 or ADD
- Input1 -> RS1 = REG[x13]
- Input2 -> Output of ALUSrc Mux
  0x0000000000000014

PC+4 takes inputs:
- Input1 -> PC
- Input2 -> 4 in decimal

Branch Adder takes inputs:
- Input1 -> PC
- Input2 -> (SignExtended(Immediate)<<1)
  = (0x0000000000000014 << 1)
  = 0x0000000000000028
  or 40 in decimal

Answer 1.5)
The Registers File takes inputs:
Read Register1 -> Instruction[19:15] = RS1 = REG[x13]
Read Register2 -> Instruction[24:20] = RS2 = REG[x12]
Write Register -> Instruction[11:7] = 10100 = REG[x20]

RegWrite -> Output from Control Unit
0 for sd instructions
Write Date -> Output of MemToReg Mux
Unknown

Since it is S-type instruction, the Write Register value REG[x20] is ignored.

Answer 2.1) Steps for an R-type instruction:

Step No.	Step	Latency (in ps)
1	PC Register Read	30
2	PC+4 Adder	150
3	Instruction Fetch from Instruction Memory	250
4	Control Unit	50
5	Register File access for RS1, RS2 and RD	150
6	Sign Extension Unit	50
7	Left Shift by 1	0
8	Branch Adder	150
9	Mux for ALUSrc	25
10	Main ALU Operation	200
11	ALU Control	50
12	AND Gate for PcSrc Mux	5
13	Mux for PCSrc	25
14	Mux for MemToReg to Write Back	25
15	Register Setup for RD	20

Notes:
1. Step 2 and Step 3 run in parallel.
2. Step 4, Step 5, Step 6 run in parallel.
3. Although Step 6 (Sign Extension Unit) is not required for R-type instruction, since we have no control signal it still calculated and the latency is therefore present.
4. Step 7 (Left Shift by 1) is just wires hardwired to the left. So no gates are used here. Hence, latency is 0. Also, similar to point 3, not required for R-type but still performed.
5. Step 8 (Branch Adder) similar to point 3.
6. Step 7 + Step 8, Step 9 + Step 10 and Step 11 run in parallel.
7. Step 12 (AND Gate) similar to point 3.
8. Data Memory is not used, hence no latency is added.
9. Step 12 + Step 13 and Step 14 + Step 15 run in parallel.

Flow:
1. Latency for Step 1 : 30
2. Latency for Step 2 and Step 3 in parallel : 250
3. Latency for Step 4, Step 5 and Step 6 in parallel : 150
4. Latency for Step 7 + Step 8, Step 9 + Step 10, Step 11 in parallel : 225
5. Latency for Step 12 + Step 13 and Step 14 + Step 15 in parallel : 45

Total Latency : 700 ps

Answer 2.2) Steps for an ld instruction (I-type)

Step No.	Step	Latency (in ps)
1	PC Register Read	30
2	PC+4 Adder	150
3	Instruction Fetch from Instruction Memory	250
4	Control Unit	50
5	Register File access for RS1 and RD	150
6	Sign Extension Unit	50
7	Left Shift by 1	0
8	Branch Adder	150
9	Mux for ALUSrc	25
10	Main ALU Operation	200
11	ALU Control	50
12	AND Gate for PcSrc Mux	5
13	Mux for PCSrc	25
14	Data Memory for load	250
15	Mux for MemToReg to Write Back	25
16	Register Setup for RD	20

Notes:
1. The only change from R-type instruction is that we need to access Data Memory here.
2. Step 12 + Step 13 and Step 14 + Step 15 run in parallel.

Flow:
1. Latency for Step 1 : 30
2. Latency for Step 2 and Step 3 in parallel : 250
3. Latency for Step 4, Step 5 and Step 6 in parallel : 150
4. Latency for Step 7 + Step 8, Step 9 + Step 10, Step 11 in parallel : 225
5. Latency for Step 12 + Step 13 and Step14 + Step 15 in parallel : 275
6. Latency for Step 15 : 20

Total Latency : 950 ps

Answer 2.3) Steps for an sd instruction (S-type)

Step No.	Step	Latency (in ps)
1	PC Register Read	30
2	PC+4 Adder	150
3	Instruction Fetch from Instruction Memory	250
4	Control Unit	50
5	Register File access for RS1 and RS2	150
6	Sign Extension Unit	50
7	Left Shift by 1	0
8	Branch Adder	150
9	Mux for ALUSrc	25
10	Main ALU Operation	200
11	ALU Control	50
12	AND Gate for PcSrc Mux	5
13	Mux for PCSrc	25
14	Data Memory for store	250

Notes:
1. The only change from I-type instruction is that we don't need to Write Back. So Steps 15 and 16 are irrelevant.
2. Step 12 + Step 13 and Step 14 run in parallel.

Flow:
1. Latency for Step 1 : 30
2. Latency for Step 2 and Step 3 in parallel : 250
3. Latency for Step 4, Step 5 and Step 6 in parallel : 150
4. Latency for Step 7 + Step 8, Step 9 + Step 10, Step 11 in parallel : 225
5. Latency for Step 12 + Step 13 and Step14 + Step 15 in parallel : 250
6. Latency for Step 15 : 20

Total Latency : 905 ps

Answer 2.4) Steps for an beq instruction (SB-type)

Step No.	Step	Latency (in ps)
1	PC Register Read	30
2	PC+4 Adder	150
3	Instruction Fetch from Instruction Memory	250
4	Control Unit	50
5	Register File access for RS1 and RS2	150
6	Sign Extension Unit	50
7	Left Shift by 1	0
8	Branch Adder	150
9	Mux for ALUSrc	25
10	Main ALU Operation	200
11	ALU Control	50
12	AND Gate for PcSrc Mux	5
13	Mux for MemToReg to Write Back	25
14	Register Setup for RD	20

Notes:
1. The only change from R-type instruction is that we don't need to Write Back. So Steps 14 and 15 are irrelevant.
2. Step 12 and Step 13 run in parallel.

Flow:
1. Latency for Step 1 : 30
2. Latency for Step 2 and Step 3 in parallel : 250
3. Latency for Step 4, Step 5 and Step 6 in parallel : 150
4. Latency for Step 7 + Step 8, Step 9 + Step 10, Step 11 in parallel : 225
5. Latency for Step 13 : 25
6. Latency for Step 14 : 20

Total Latency : 705 ps

Answer 2.5) If the I-type instruction is a load instruction which requires access to Data Memory, the latency would be **950 ps.**
Else, for all other I-type instructions like and immediate, or immediate, shift immediate, it will take the same latency as R- type instruction, i.e. **700 ps.**

Answer 2.6) The minimum clock cycle period for a non-pipelined processor would be that of load instructions, i.e. **950 ps.**
If we make the processor pipelined, we could use a minimum clock cycle of **250 ps** which is the latency for memory accesses, be it instruction memory or data memory.

Answer 3.a)
Let the total number of instructions be n
For the instruction mix given,
New Execution Time
= 0.52*n*700 + 0.25*n*950 + 0.11*n*905 + 0.12*n*685
= 364n + 237.5n + 99.55n + 82.2n
= 783.25n
Old Execution Time = 950*n
Speedup = 950/783.25 = 1.21 times

Answer 3.b)
New ALU latency = 200 + 300 = 500ps
Thereby increasing the latency by 300 for all R type, ld, sd and beq instructions.

3.b1) Clock cycle $time_{old}$ = 950 ps
Clock cycle $time_{new}$ = 950 + 300 = 1250ps

3.b2) Clock cycle $time_{old}$ = 950*n
Clock cycle $time_{new}$ = 1250*n * 0.95 (reducing 5% instructions)

Speedup = Old/new
= 950 / 1250 * 0.95
= 0.8

Hence, the new processor will be 20% slower than the old one.

3.b3) The improved performance can be achieved if (t is ALU latency) –
(950+t)*0.95*n < 950n
t < 50 ps

Hence, to get the slowest ALU, it needs to be less than 50ps additionally.
Therefore, the new ALU time would be 250ps.

Answer 3c)
New Register File time = 160 ps
Latency for R-type instruction = 700 + 10 = 710 ps
Latency for ld instruction = 950 + 10 = 960 ps
Latency for sd instruction = 905 + 10 = 915 ps
Latency for beq instruction = 705 + 10 = 715 ps

3.c1)
Total instructions for the new processor
= 0.52*n + 0.25*0.88*n + 0.11*0.88*n + 0.12*n
= 0.52n + 0.22n + 0.0968n + 0.12n
= 0.9568n
Speedup = 950/(960*0.9568) = 1.03 times

3.c2)
Component Mix for Old Processor (not 1):
MUX – 3
Adder – 2

Total Cost of Old Processor
= 1*1000 + 1*200 + 3*10 + 1*100 + 2*30 + 1*2000 + 1*5 +
1*100 + 1*1 + 1*500
= 3996

For the new processor, cost of register file becomes 400.
Total Cost of New Processor
= 3996+200
= 4196

Therefore, the change in cost = 4196/3996= 1.05% i.e., 5% improvement.
Change in performance from 3.c1) is 3%

3.c3)
Cost/Performance ratio = 5/3 = 1.66
If 1.66 ratio is significant than it makes sense to increase the number of registers. It depends on the marginal utility of the performance gain.

Answer 4.1)
I-Mem/D-Mem latency = 250 ps
ALU latency = 200 ps
Additionally, no Mux would be used before this = 25ps.
Since, no 2 instructions needs both to be sequentially executed, we can parallelize them.

(PC Read reg = 30ps) + (instruction memory = 250ps) + (read reg file = 150ps) + (read data from Data mem = 250ps) + (Mux for writeback = 25ps) + (write back for reg - 20ps)

psTotal (ld)= 725ps

(PC Read reg = 30ps) + (instruction memory = 250ps) + (read reg file - 150ps) + (write data from reg to Data mem = 250ps)

psTotal (sd)= 680ps

Answer 4.2)
The number of ld instructions is 0.25*n
The number of sd instructions is 0.11*n
Hence, we have additional R-type add instructions 0.36*n
Total Instructions = 1.36n
Execution Time of New Processor = 1.36*n*725 = 986*n
Speedup = 950/986 = 0.963
Hence, the new processor would be slower by approximately 3.65%

Answer 4.3)
The clock cycle time is usually maximum for the load instructions in a processor. A new CPU should be opted for only when there are **lesser** ld and sd instructions.

Answer 4.4)
As per the above answer, depending on how often the ld/sd instruction occur. The improvement to cycle time is 22% by eliminating the ALU op for ld/sd. Although, if ld/sd occur more than 22% of instructions then it is better not to get new CPU.

Answer 5)
Load with Increment adding requires

i. New Functional blocks
   To determine the base address of the data to be loaded, we combine the offset and RS1 values using the ALU. As an alternative, we would add RS1 and RS2 in this case to obtain the base address of the data to be loaded.
   Hence, no new functional block is required.

ii. Modification of existing functional blocks
   According to the current load instructions, the ALUSrc signal from the Control Unit directs the ALUSrc Mux to obtain the data from the sign extended immediate value. We can **update the Control Unit to select RS2 rather than sign extended immediate value for lwi.d instructions depending on the opcode of the lwi.d instruction**.

iii. New Data path
   The write back operation is then continued once the ALU Unit has calculated RS1+RS2, the Data Memory has retrieved the value at address RS1+RS2, and so forth we continue with the write back process. Hence, **no need for new data path as data memory is already connected sequentially to ALU unit.**

Answer 6)

6.1)
We are aware that the clock cycle time in a non-pipelined processor might be influenced by the ld instruction's latency. All five phases would be necessary for the ld instruction: Instructions are fetched, decoded, executed, and written back using data memory.

Clock Cycle Time (Non Pipelined)
= 250 + 350 + 150 + 300 + 200
= 1250 ps

Herein, the maximum clock cycle time would be choosen in pipelined processors i.e., 350 ps.

6.2)
While using ld in both pipelined and non-pipelined cases, it would require the same time i.e., 1250 ps due to pipelining affecting the throughput of the instructions and not just any individual latency.

6.3)
It is advisable to split the stage having maximum latency (ID) into 175ps each. Hence, this would reduce the pipelined processors' clock cycle time to 300 ps.
Although, for non – pipelined processor it would still be 1250 ps as the sum would remain constant.

6.4)
For the Load and Store instructions, data memory is used. Hence, the utilization comes out to be 20% + 15% = 35%
6.5)
The regWrite port is mainly used for calculation of R type instructions coming from the ALU along with the data retrieved from Data memory during Load instruction. Hence, the utilization comes out to be 20% + 45% = 65%

Answer 7)

In case of no stalls, the no of clock required would be as follows –

i. 1 instruction: k
ii. 2 instructions: k +1
iii. 3 instructions: k + 2

Therefore, for n instructions, the clock cycles required would come out to be: k + (n-1)
where,

k - stages
n - instructions

Answer 8a)

Value of register x11: 11
Value of register x12: 12

Code -

```
addi x11, x12, 5
add x13, x11, x12
addi x14, x11, 15
```

(Above Given in Q8a)

Order of stages execution is defined below –

IF	ID	EX	MEM	WB

If the NOP instructions aren't added by the programmer to mitigate the data hazards.

i. The first instruction would calculate x11 to be 22+5 = 27 only in the 5th cycle.

ii. The second instruction would still use x11 as 11 and would calculate x13 to be 11+22 = 33

iii. The third instruction's ID stage would still not be in the same cycle as the WB stage of the first instruction. Hence, it would also use x11 as 11 and would calculate x14 to be 11+15 = 26

x13 = 33
x14 = 26

Answer 8b)
Value of register x11: 11
Value of register x12: 12

Code –
addi x11, x12, 5
add x13, x11, x12
addi x14, x11, 15
add x15, x11, x11

ID is able to fetch the register value written in the same cycle in WB stage.

(Above Given in Q8b)

IF	ID	EX	MEM	WB

Here, the fourth instruction's ID stage would be in the same cycle as first instruction's WB stage. Hence, it would get the update x11 value as 22+5 = 27.
x15 = 27 + 27 = 54

Answer 8c)

The principal instruction works out x11 in the fifth cycle.
The subsequent instruction requires x11 in the last part of the second cycle.
This implies there ought to be two NOP instructions for no data hazard.
Post 2nd NOP
X11 -> WB
X13 -> ID

Then, the third instruction pursues the subsequent instruction what's more, since the NOPs are as of now added, it can execute accurately.

The fourth instruction requires x13 and x12. x13 is determined in the subsequent instruction and since we just have a hole of 1 instruction, we want to add a NOP for the ID of fourth instruction furthermore, WB of second instruction to match up.

addi x11, x12, 5
NOP

NOP
add x13, x11, x12
addi x14, x11, 15
NOP
add x15, x13, x12