For the following ILP example, calculate CPI for only the fetch stage in each case: design a super scalar based on this with load/store, branch, and ALU units that requires a minimum number of total units, so that this program will take the smallest number of cycles to finish.

0: LDUR
1: ADD
2: LSR
3: AND
4: EORI
5: CBZ
6: SUBI
7: CMOVZ
8: STURB
9: B.LT
We are given the following assembly code:

    CBZ  X1, SKIP
    ADD  X1, X5, X31

SKIP:
    STUR X1, [X2, #0]
    ADD  X1, X3, X4
    BR   X1

a) Change this code so that a 2-way VLIW with 1 branch/ALU and 1 load/store/ALU runs as fast as possible.
b) Schedule this new code.

You may change anything so long as the result stored in memory and the final location that is branched to are the same. Assume no delay slots and that all instructions parallel to a branch will execute.
Given the following cache structure, determine the average access times of each level.

<table>
<thead>
<tr>
<th>LEVEL</th>
<th>HIT TIME</th>
<th>HIT RATE</th>
<th>ACCESS TIME</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1</td>
<td>1 cycle</td>
<td>98%</td>
<td></td>
</tr>
<tr>
<td>L2</td>
<td>20 cycles</td>
<td>80%</td>
<td></td>
</tr>
<tr>
<td>MM</td>
<td>100 cycles</td>
<td>97%</td>
<td></td>
</tr>
<tr>
<td>DISK</td>
<td>10,000 cycles</td>
<td>100%</td>
<td></td>
</tr>
</tbody>
</table>

Additionally, if you can speed up any levels hit time by 30%, which is the best to speed up?
Draw the constraint graph for the code below and label the cause of the constraint:

LOOP:

LDUR X0, [X31, #10]
ADD X1, X31, X31
STUR X2, [X1, #0]
SUB X3, X2, X0
ADD X4, X5, X6
STUR X4, [X7, #8]
CBZ X4, LOOP
A 2-bit predictor will be correct — \%. 

A 1 bit?

\[
\text{while}(1)\{
    \text{if} \ (i \leq 4) \ \text{counter}++;
    i = (i+1) \mod 5;
\}
\]
If given a total size of 64 bytes cache with 8-byte blocks, determine what type of cache it is based on the following cache miss/hit series.

24 miss
32 miss
80 miss
24 hit
0 miss
48 miss
40 miss
32 hit
0 hit
64 miss
72 miss
8 miss
0 hit
80 hit
How would the following code be scheduled on a 2-way VLIW with no delay slot, a 2-way superscalar with no miss, and a 2-way superscalar with 2-cycle miss.

1: LDUR X0, [X3, #0]
2: ADD X1, X0, X2
3: STUR X2, [X3, #0]
4: ANDI X2, X5, #1
5: ORR X6, X5, X4
6: ADDI X7, X6, #2
7: EORI X8, X4, #3
8: LSL X8, X8, #2
9: STUR X8, [X4, #16]
10: ADDI X9, X5, #1

<table>
<thead>
<tr>
<th>VLIW</th>
<th>ALU/Load/Store</th>
<th>ALU/BR</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Superscalar with no miss</th>
<th>ALU/Load/Store</th>
<th>ALU/BR</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>2-cycle miss</th>
<th>ALU/Load/Store</th>
<th>ALU/BR</th>
</tr>
</thead>
</table>
Will a 4-way superscalar machine always be faster than a 2-way superscalar machine? Why or why not?

Will a 4-way superscalar machine always be faster than a 4-way VLIW machine? Why or why not?
Create the simplest 5 stage pipelined CPU to do the following 4 instructions:

ADD
LDUR
STUR
CBZ
Consider this series of address references:

8, 12, 5, 8, 12, 5, 8, 12, 5, 8, 12, 6, 8, 12

For each address, show the binary value, Cache Tag, Cache Index, and Byte Select for each cache. Show final cache contents for a fully associative cache with 8 byte blocks and total size of 32 bytes.
Question:

Write assembly code for this without using branches:

```java
for (int i = 0; i < array.length; i++) {
    if (array[i] < 0) {
        array[i] = i;
    }
}
```
Using a 14 bit address, find the total amount of memory needed to implement the cache for a two-way set associative, 4B block.