Create a single-cycle CPU that can perform following two instructions. The CPU should be as simple as possible.

STUR Rd, [Rn, DAddr9];
LDUR Rd, [Rn, DAddr9];

a. Draw the datapath of your CPU.

b. Can these two instructions complete in one cycle? If they can, update your datapath and show the control signal setting. If they cannot, show your reasons.
What would the following C++ code be in memory? Assume we start using memory at 1000.

```cpp
int var = 100;
int *ptr1;
int **ptr2;

ptr1 = &var;
ptr2 = &ptr1;
```
Consider a computer executing the following mix of instructions

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Frequency</th>
<th>Clock Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>40%</td>
<td>1</td>
</tr>
<tr>
<td>LOAD</td>
<td>20%</td>
<td>5</td>
</tr>
<tr>
<td>STORE</td>
<td>20%</td>
<td>5</td>
</tr>
<tr>
<td>BRANCH</td>
<td>10%</td>
<td>2</td>
</tr>
</tbody>
</table>

What is the average CPI?

\[
\text{CPI} = \frac{(0.4 \times 1) + (1.0 \times 5) + (1.0 \times 5) + (0.2 \times 2)}{4} = 2.6
\]

Should \(\leq 100\%\)
Translate the Java code to ARM assembly code. Use a minimum number of instructions. Assume that v is an array of integers and the length of v (v.length) is stored in register x0. You can use as many register as you want but keep it simple.

```java
CompStore (int [ ] v){
    for(int k = 0; k < v.length; k++) {
        v[k] = k + v[k-1] - v[k];
    }
}
```

```
ADDI x2, x3, 0  // k=0

Loop:  
LSL x3, x2, #3  // k<<8
ADD x3, x3  // &v[k]

STUR x2, [x3, #03] // v[k] = k
ADDI x2, x2, #1  // k++
CMP x2, x0  // k< v.length
BLT Loop
```
Bad Things happened, and now our 5-stage pipelined CPU with forwarding is running the following instructions:

\[
\text{ADDI } x_0, x_{31}, \#31 \\
\text{LDUR } x_1, [x_0, \#1] \\
\text{LDUR } x_2, [x_0, \#9] \\
\text{ADDI } x_3, x_2, x_1 \\
\text{ADD } x_{31}, x_{31}, x_{31}
\]

What are the resulting values in registers \( x_0, x_1, x_2, \) and \( x_3 \)?

You may leave answers as a sum of memory values and constants.

Example: \( x_0 = \text{mem}[16] + \text{mem}[8] + 29 \)

\[
\begin{align*}
x_0 &: 31 \\
x_1 &: \text{mem}[323] \\
x_2 &: \text{mem}[40] \\
x_3 &: 40 + \text{mem}[32]
\end{align*}
\]
Consider the pipelined CPU that has branch and load delay slots, forwarding logic, and accelerated branches. For the following code, explain what is happening in each stage of the pipelined processor during cycle 5.

```
1. 2. 3. 4. 5.
S  DEC  EX  MEM  WB
```

- **LDUR**: If DEC, IF EX MEM
- **ADD**: If DEC, IF EX
- **LDUR**: IF Dec
- **CBZ**: If fetching CBZ

If the branch is taken, the program counter is not updated by the jump. If the branch is not taken, the program counter is advanced.

```
W  T.  T.  T.  T.
```

- **ADD**: If DEC, EX
- **STUR**: If DEC, WB
- **LDUR**: X3, [X31, #32]
- **ADD**: X1, X2, XZ
- **LDUR**: X3, [X31, #100]

```
```

**Loop:**