The code fragment below, in C source and assembler forms, is referred to in the problems below.

```c
for(i=0; i<1000; i++) if( s[i].type == 0 )
    suma += s[i].score; else sumb+=s[i].score;

! r3 initialized to address of first element.
    add r1, r0, r0  ! i=0
LOOP:
    slti r2, r1, #1000  ! r2 = 1 if r1 < 1000, otherwise r2 = 0.
    beqz r2, DONE
    lw r4, 0(r3)
    ld f0, 16(r3)
    bneq r4, SUMB  ! Taken half the time.
    addd f2, f2, f0
    j NEXT
SUMB:
    addd f4, f4, f0
NEXT:
    addi r3, r3, #64  ! Size of element is 64 bytes.
    addi r1, r1, #1  ! Increment loop index.
    j LOOP
DONE:
```

**Problem 1:** Determine the static and dynamic instruction count for the DLX program above. The branch that tests r4 will be taken half the time.

The dynamic count for each instruction is shown in the first column:

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Dynamic Count</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>! r3 initialized to address of first element.</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>add r1, r0, r0  ! i=0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>LOOP:</td>
<td>1000</td>
<td>slti r2, r1, #1000  ! r2 = 1 if r1 &lt; 1000, otherwise r2 = 0.</td>
</tr>
<tr>
<td>1001</td>
<td>beqz r2, DONE</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>lw r4, 0(r3)</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>ld f0, 16(r3)</td>
<td></td>
</tr>
<tr>
<td>SUMB:</td>
<td>500</td>
<td>bneq r4, SUMB  ! Taken half the time.</td>
</tr>
<tr>
<td>NEXT:</td>
<td>500</td>
<td>addd f2, f2, f0</td>
</tr>
<tr>
<td>500</td>
<td>j NEXT</td>
<td></td>
</tr>
<tr>
<td>DONE:</td>
<td>500</td>
<td>addd f4, f4, f0</td>
</tr>
</tbody>
</table>

Static: 12 instructions. Dynamic: 9,503 instructions (totaling dynamic counts above).
**Problem 2:** Suppose the program runs for 1 millisecond on a system with a 10 MHz clock. Assuming no cache misses (an assumption that will be made for most of these problems), what is the average CPI?

Answer: 
\[
CPI = \frac{1 \text{ ms} \times 10 \text{ MHz}}{950 \text{ inst}} = \frac{10000 \text{ cycles}}{9503 \text{ inst}} = 1.0523 \text{ CPI}
\]

**Problem 3:** Divide the instructions into two classes: floating-point and others. (The floating-point instructions include the `add` and `ld` instructions.) Suppose on implementation A the CPI of floating-point instructions, CPI\text{fp}, is twice the CPI of the other instructions, CPI\text{other}. If implementation A uses a 10 MHz clock and runs the program in 1 millisecond (like the previous problem), what would the CPIs be? Implementation B is the same as implementation A except floating-point instructions have an average CPI that is 3 times the other instructions. Estimate how long it will take to run the program on implementation B using a 10 MHz clock.

Let \( t_A \) denote the execution time on implementation A (which can be expressed in cycles or seconds).

\[
t_A = \frac{CPI\text{fp} \times C\text{fp} + CPI\text{other} \times C\text{other}}{C\text{fp} + C\text{other}} = \frac{2 \times CPI\text{other} \times C\text{fp} + CPI\text{other} \times C\text{other}}{2 \times C\text{fp} + C\text{other}}
\]

Solving for CPI\text{other}:

\[
CPI\text{other} = \frac{t_A}{2 \times C\text{fp} + C\text{other}} = \frac{10000 \text{ cycles}}{2 \times 2000 + 7503} = 0.8692 \text{ CPI}
\]

Then CPI\text{fp} = 2 \times CPI\text{other} = 1.7387 \text{ CPI}. Let \( t_B \) denote the execution time estimate for implementation B. Then

\[
t_B = CPI\text{other} \times (3 \times C\text{fp} + C\text{other}) = 0.8693(3 \times 2000 + 7503)
\]
\[
= 11738.7 \text{ cycles} = 1.17387 \text{ ms}
\]

**Problem 4:** Suppose that an implementation executed instructions one after another with no overlapping and no gaps between instructions. If each instruction took five cycles to execute and the clock frequency was 10 MHz, how long would program execution take?

It would take \( 5 \times 9503 = 47515 \text{ cycles} = 4.7515 \text{ ms} \).
**Problem 5:** Suppose, somehow, a load double and load word instruction using scaled addressing were added to DLX. The assembler syntax is similar to the one in table 2.5 of the text, except a displacement is included at the end. For example, the execution of `ld f0, 10(r20)[r30]40` will load f0 (and f1) with the contents of memory at address $10 + r20 + r30 \times 40$. Rewrite the program above using the new instruction.

! r3 initialized to address of first element.
    add r1, r0, r0 ! i=0
LOOP:
    slti r2, r1, #1000 ! r2 = 1 if r1 < 1000, otherwise r2 = 0.
    beqz r2, DONE
    lw r4, 0(r3)[r1]64
    ld f0, 16(r3)[r1]64
    bneq r4, SUMB ! Taken half the time.
    add f2, f2, f0
    j NEXT
SUMB:
    add f4, f4, f0
NEXT:
    ! Note that r3 is no longer changed.
    addi r1, r1, #1 ! Increment loop index.
    j LOOP
DONE: