The SPARC assembly language program below is used in the problems that follow. SPARC register names are \( %g0-%g7 \), \( %i0-%i7 \), \( %l0-%l7 \), and \( %o0-%o7 \); and \( %g0 \) is a zero register (like \( r0 \) in DLX). The destination for arithmetic, logical, and load instructions is the rightmost register (\( \text{add} \ %11, \%12, \%13 \) means \( \%13 = \%11 + \%12 \)). SPARC uses a condition code register and special condition-code-setting instructions for branches. Branches include a delay slot.

```assembly
LOOP:
  ld [%l1], %l2 ! Load l2 = MEM[ l1 ]
  addcc %l2, %g0, %g0 ! g0 = g0 + l2. Sets cond. codes. Note: g0 is zero reg.
  be DONE ! Branch if result zero.
  nop ! Fill delay slot with nop.
  add %l6, %l2, %l6 ! l6 = l6 + l2
  andcc %l3, 1, %g0 ! g0=1&l 3 . Sets cond. codes. Note: g0 is zero reg.
  be SKIP1
  nop
  add %l4, 1, %l4

SKIP1:
  subcc %l3, 1000, %g0
  bpos SKIP2 ! Branch if >= 0;
  nop
  add %l4, %l3, %l4

SKIP2:
  andcc %l3, 1, %g0
  be SKIP3
  nop
  add %l4, %l4, %l4

SKIP3:
  add %l1, 4, %l1
  ba LOOP ! Branch always. (Jump.)
  nop

DONE:
```

**Problem 1:** An execution of the code above on a SPARC implementation takes 1000 cycles. The dynamic instruction count is \( IC_{\text{all}} \) of which \( IC_{\text{nop}} \) instructions are \texttt{nop}'s. Consider two ways of computing CPI:

\[
\text{CPI}_A = \frac{t}{IC_{\text{all}}}, \quad \text{and} \quad \text{CPI}_B = \frac{t}{IC_{\text{all}} - IC_{\text{nop}}},
\]

where \( t \) is the execution time in cycles. Which is better? Justify your answer; an argument for either formula can be correct.

\( \text{CPI}_A \) is better because it measures how efficiently a processor executes instructions, including \texttt{nop} instructions which are part of the code.
**Problem 2:** SPARC branches have a one-instruction delay slot, in the code above they are filled with *nop*’s. Re-write the code filling as many slots with useful instructions as possible, reducing the number of instructions in the program.

**Solution:**

```assembly
1d [%l1], %l2 ! Load 12 = MEM[ l1 ]

LOOP:
addcc %l2, %g0, %g0 ! g0 = g0 + 12. Sets cond. codes. Note: g0 is zero reg.
be DONE ! Branch if result zero.
andcc %l3, 1, %g0 ! g0 = 1 & l3. Sets cond. codes. Note: g0 is zero reg.
add %l6, %l2, %l6 ! l6 = l6 + l2
be SKIP1
subcc %l3, 1000, %g0
add %l4, 1, %l4

SKIP1:
bpos SKIP2 ! Branch if >= 0;
andcc %l13, 1, %g0
add %l4, %l3, %l4

SKIP2:
be SKIP3
add %l11, 4, %l11
add %l4, %l14, %l14

SKIP3:
b beqz r10, SKIP1 ! r10 computed before SKIP1.
add r4, r4, #1
beqz r0, LOOP
DONE:
```

**Problem 3:** Re-write the program in DLX, taking advantage of DLX’s use of general purpose registers for specifying branch conditions.

**Solution:**

```
LOOP:
lw r2, 0(r1)
beqz r2, DONE
add r6, r2, r6
andi r10, r3, #1
beqz r10, SKIP1
addi r4, r4, #1

SKIP1:
sgei r11, r3, #1000
bneq r11, SKIP2
add r4, r3, r4

SKIP2:
beqz r10, SKIP3 ! r10 computed before SKIP1.
add r4, r4, r4

SKIP3:
addi r1, r1, #4
beqz r0, LOOP
DONE:
```
Problem 4: The program below executes on the DLX implementation shown below. The comments show the results of the xori, or, and lw instructions.

Initially, r1=11, r2=22, r3=33, etc.
addi r0, r0, #0
addi r0, r0, #0
addi r0, r0, #0
addi r0, r0, #0
addi r0, r0, #0
START: ! START = 0x50
xori r1, r9, #7 !99 $7 = 100
or r2, r3, r4 !33 or 44 = 45
lw r5, 9(r6) !Mem[9+66]=42
sw 10(r7), r8
addi r0, r0, #0
addi r0, r0, #0
addi r0, r0, #0
addi r0, r0, #0
addi r0, r0, #0

The table below shows the contents of pipeline registers and changes to architecturally visible registers r1-r31 over time. Cycle zero is the time that xori is in instruction fetch. The first two columns are completed, continue filling the table up until the sw instruction finishes writeback. Ignore values which are not used and which depend on the func field of type-R instructions. Values which are not used and don’t depend on the func field should be shown. The output of the data memory is zero when a store or no memory operation is performed. The row labeled “Reg. Chng.” shows a new register value that is available at the beginning of the cycle. If no register value is written leave the entry blank.

<table>
<thead>
<tr>
<th>Cycle</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>0x50</td>
<td>0x54</td>
<td>0x58</td>
<td>0x5c</td>
<td>0x60</td>
<td>0x64</td>
<td>0x68</td>
<td>0x6c</td>
<td>0x70</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>IF/ID.IR</td>
<td>addi</td>
<td>xori</td>
<td>or</td>
<td>lw</td>
<td>sw</td>
<td>addi</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reg. Chng.</td>
<td>r0 ← 0</td>
<td>r0 ← 0</td>
<td>r0 ← 0</td>
<td>r0 ← 0</td>
<td>r0 ← 0</td>
<td>r1 ← 10</td>
<td>r2 ← 45</td>
<td>r5 ← 42</td>
<td>X</td>
<td>r0 ← 0</td>
<td>...</td>
</tr>
<tr>
<td>ID/EX.IR</td>
<td>addi</td>
<td>addi</td>
<td>xori</td>
<td>or</td>
<td>lw</td>
<td>sw</td>
<td>addi</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID/EX.A</td>
<td>0</td>
<td>0</td>
<td>99</td>
<td>33</td>
<td>66</td>
<td>77</td>
<td>0</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID/EX.B</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>44</td>
<td>55</td>
<td>88</td>
<td>0</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ID/EX.IMM</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>X</td>
<td>9</td>
<td>10</td>
<td>0</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX/MEM.IR</td>
<td>addi</td>
<td>addi</td>
<td>addi</td>
<td>xori</td>
<td>or</td>
<td>lw</td>
<td>sw</td>
<td>addi</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX/MEM.ALU</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>45</td>
<td>75</td>
<td>87</td>
<td>0</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EX/MEM.B</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>44</td>
<td>55</td>
<td>88</td>
<td>0</td>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEM/WB.IR</td>
<td>addi</td>
<td>addi</td>
<td>addi</td>
<td>addi</td>
<td>xori</td>
<td>or</td>
<td>lw</td>
<td>sw</td>
<td>addi</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>MEM/WB.ALU</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>100</td>
<td>45</td>
<td>75</td>
<td>87</td>
<td>0</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>MEM/WB.MD</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>42</td>
<td>0</td>
<td>0</td>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3