EE 4720 **Homework 2** **Due: 19 February 1999**

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 (`add %l1,%l2,%l3` means `%l3=%l1+%l2`). SPARC uses a condition code register and special condition-code-setting instructions for branches. Branches include a delay slot.

```
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 & l3.  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 $\mathrm{IC}_{\mathrm{all}}$ of which $\mathrm{IC}_{\mathrm{nop}}$ instructions are `nop`'s. Consider two ways of computing CPI:

$$\mathrm{CPI}_{\mathrm{A}} = \frac{t}{\mathrm{IC}_{\mathrm{all}}} \qquad \text{and} \qquad \mathrm{CPI}_{\mathrm{B}} = \frac{t}{\mathrm{IC}_{\mathrm{all}} - \mathrm{IC}_{\mathrm{nop}}},$$

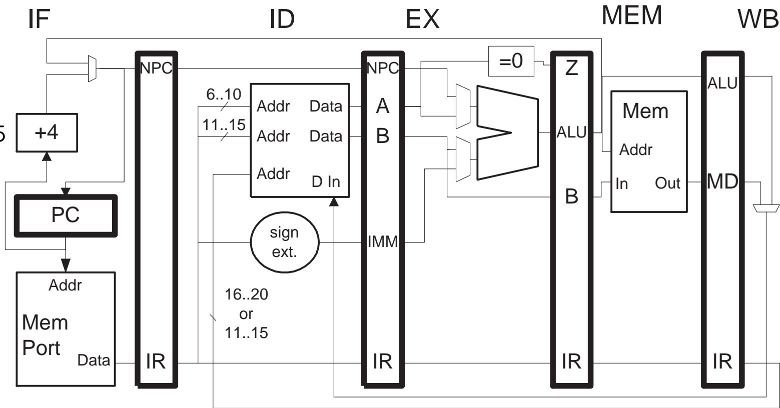
where t is the execution time in cycles. Which is better? Justify your answer; an argument for either formula can be correct.

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.

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

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.

Cycle	0	1	2	3	4	5	6	7	8	9	10
PC	0X50	0X54									
IF/ID.IR	addi	xori									
Reg. Chng.	r0 ← 0	r0 ← 0									
ID/EX.IR	addi	addi									
ID/EX.A	0	0									
ID/EX.B	0	0									
ID/EX.IMM	0	0									
EX/MEM.IR	addi	addi									
EX/MEM.ALU	0	0									
EX/MEM.B	0	0									
MEM/WB.IR	addi	addi									
MEM/WB.ALU	0	0									
MEM/WB.MD	0	0									