# INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
## Class Test 1 - 2023

**Date:** 06-02-23 **Timing:** 3 pm - 4:30 pm **Place:** NR223 **Total Marks =** 30
**Subject No :** CS60003 **HIGH PERFORMANCE COMPUTER ARCHITECTURE**
**Department/Centre/School :** Computer Science and Engineering
**Answer all questions.** In case of reasonable doubt, make assumptions and state them upfront. Marks will be deducted for claims without proper reasoning. Write your answers (with all analysis, justification, calculation etc) within the allotted time.

---

1. A quad-core processor could speed up a computer by a factor of 4 but this rarely happens. Use Amdahl's Law to compute the percentage of program execution that needs to be distributed across all 4 cores to achieve an overall speedup of 3.

   **[5]**

2. A benchmark has a breakdown of the following:

   39% loads, 12% stores, 28% ALU operations other than multiplies/divides, 6% multiplies, 2% divides, 10% conditional branches, and 3% unconditional branches.

   A processor has a CPI of 5 for loads/stores, 3 for all ALU operations other than multiplies/divides, 6 for multiplies, 15 for divides, and 4 for branches. We are considering making one of several enhancements to the ALU. Which should we make?

   a. Improving the condition tester so that conditional branch CPI is reduced to 2.
   b. Improving the multiplier so that the CPI for multiplies is reduced to 3.
   c. Improving the divider so that the CPI for divides is reduced to 7.
   d. Improving the ALU so non-multiply/divide operations have a CPI of 2.

   **[10]**

3. Consider the two functionally identical code snippets given below:

| Version 1 | Version 2 |
|---|---|
| `loop: lw r0 r3 100` | `loop: lw r0 r3 100` |
| `nor r3 r4 r5` | `nor r3 r4 r5` |
| `nor r3 r4 r6` | `add r5 r0 r7` |
| `add r5 r0 r7` | `nor r3 r4 r6` |
| `beq r0 r0 loop` | `beq r0 r0 loop` |

   For both code snippets, list out all data dependencies that cause data hazards in the 5-stage pipeline, as discussed in class. Consider only 1 iteration of the loop.

   **[5]**

**[Please Turn over]**

4. Consider the following C code and its corresponding assembly. Assume it is executed on a pipelined data-path with branch speculation discussed in class.

```
int i, j, x = 0;
for (i=0; i != 30; i++) {
    for (j=0; j != 2; j++) {
        x++;
    }
}
```

```
          lw   r1 cnt1
          lw   r2 cnt2
          lw   r3  one
outer:    nop
          beq  r0  r1   exit
  B1
          nop
          lw   r2 cnt2
inner:    beq  r0  r2   done
  B2
          add  r4  r3    r4
          add  r2  r3    r2
          beq  r0  r0  inner
  B3
 done:    add  r1  r3    r1
          beq  r0  r0  outer
  B4
 exit:   halt
 cnt1   .fill 30
 cnt2   .fill  2
 one    .fill -1
```

a. How many branch instructions are executed for a single run of the program?

B1: _____ B2: _____ B3: _____ B4: _____

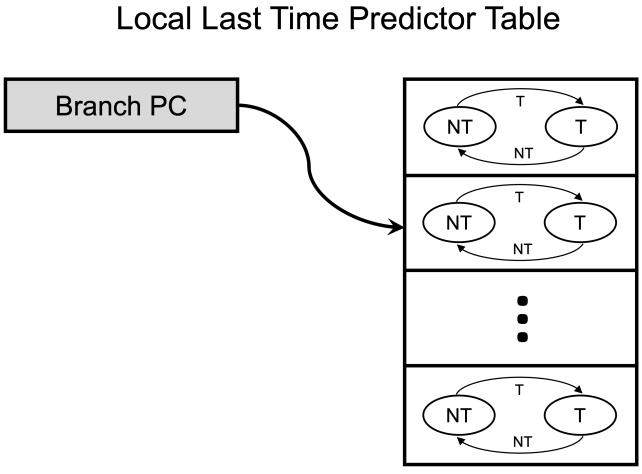
b. How many branches are predicted correctly if we predict Always-Not-Taken?

B1: _____ B2: _____ B3: _____ B4: _____

c. How many branches are predicted correctly if we use local last time 1-bit predictor?

B1: _____ B2: _____ B3: _____ B4: _____

Local 1-bit Predictor: In this prediction scheme, the PC of a branch instruction is used to index into a table. Each table entry for a branch is a 1-bit predictor, which predicts the same outcome as last time for that branch. Assume that the entries are initialized to Not Taken.




**[10]**