Branch Delay Slots

1. The ELCC MIPS compiler was used to generate the MIPS assembly language code shown below. The code contains many examples of branch delay slots. **Research** and **describe** the concept of branch delay slots in scalar pipelines. What is the difference between a branch delay slot filled with a stall versus a usable instruction and which type does this code segment illustrate? Do branch delay slots remove control hazards? Do branch delay slots improve performance?

```
main:

     addiu     $sp, $sp, -16
     sw        $fp, 12($sp)
     move      $fp, $sp
     sw        $zero, 8($fp)
     addiu     $1, $zero, 10
     sw        $1, 4($fp)
     sw        $zero, 0($fp)
     j         $BB0_1
     nop
$BB0_1:
     lw        $1, 4($fp)
     lw        $2, 0($fp)
     addu      $1, $2, $1
     sw        $1, 0($fp)
     lw        $1, 4($fp)
     addiu     $1, $1, -1
     sw        $1, 4($fp)
     j         $BB0_2
     nop
$BB0_2:
     lw        $1, 4($fp)
     bnez      $1, $BB0_1
     nop
     j         $BB0_4
     nop
$BB0_4:
     lw        $2, 0($fp)
     move      $sp, $fp
     lw        $fp, 12($sp)
     addiu     $sp, $sp, 16
     jr        $ra
     nop
```

Branch Predictors

2. **Consider** a MIPS processor design that uses a four-state branch predictor with states strongly not taken (SN), weakly not taken (WN), weakly taken (WT), and strongly taken (ST). The machine **resets** into state SN and moves progressively toward ST with each taken branch. Similarly, the machine moves in the reverse direction toward SNT with each non-taken branch. The machine saturates in SN and ST as long as N and T branch results respectively continue to calculate in the execution circuit. The name of the state is the predicted PC multiplexer path (NT or T).

   a. **Draw** the state machine.
   b. **Determine** the state of the prediction state machine if the **actual branch behavior as calculated by the branch execution circuit** after reset is TTTTT_N_TTTTTTT_N_TT_NNNN_TTT_N_T_N_TT_NN_TTT_N_T_N_T
   c. The prediction result of the machine is used to select either the PC+4 or branch target address (which is calculated in the IF state by moving sign-extension of the imm32 and the left

3. **Consider** a MIPS processor that uses the standard 2-bit saturation counter described in problem 2. Suppose the branch penalty to refill the pipeline with useful instructions is 2 cycles (on the clock tick, IF/ID bubbles, new PC moves correct instruction into IF, on the next clock tick the correct instruction moves to decode giving useful information in the circuits again). **Assume** the predictor has saturated into the ST state. How many penalty cycles get inserted when the **actual branch behavior as calculated by the branch execution circuit** is TTTTTT_NNN_TT_NN_TTTTTTT_NN_T

Predicated Instructions

Conditional, or predicated instructions, are instructions that execute only if a condition is met. Many instruction sets include basic predicated instructions to handle hazards that arise because of very common programming structures. For example, the code segment:

```
if (A == B) then
{
  X = 12;
}
else
{
 X = Z;
 Y = J;
}
```

is a very common programming structure in C. The if-then-else requires branch statements in the MIPS assembly language and introduces control hazards to the pipeline flight-plan. Predicated instructions are one compiler-time technique to remove branch hazards. Common programming structures are identified by the compiler at compile time and replaced with predicated instructions that execute only if their condition is true. In the examples above, the addition of two predicated move (also called a conditional move) instructions to the MIPS instruction set could eliminate all branches from the flight plan. This new MIPS assembly language instruction might look like this:

FIELDS	ASSEMBLY EXAMPLE	BEHAVIOR
cmovz rd,rs,rt	cmovz $s0,$s5,$t3	R[s0] = R[s5] if $t3 = 0
cmovnz rd,rs,rt	cmovnz $s0, $s5, $t3	R[s0] = R[s5] if $t3 != 0

Of course, the value in register t3 is the conditional evaluation that controls the if-then-else statement and must be created by instructions executing before the conditional move. In this case, a subtraction of A and B would place a result in $t3 before entry into the if-then-else.

4. **Translate** the C program to the standard MIPS assembly language used in CE2930. **Do not include** the new predicated instructions. **Choose** appropriate saved registers ($s0-$s7) for variable names and temporary registers ($t0-$t9) for any intermediate calculation.
5. **Rework** problem 4 using the new predicated instructions.
6. **Compare** the performance of the code segments in problems 4 and 5 for both of these cases: A = B and A != B.

DUE DATE: These problems are due on printed paper by 5 p.m. on Friday of week 5.