

**ECE 366—Fall 2001, Instructor: Prof. Shantanu Dutt**

**Homework 3: Due Mon, Nov 19**

1. (a) In class, we saw the design of the P,G generation logic and the corresponding carry-generation logic for 4-bit chunks of an adder. Design the P,G and carry-generation logic for 3-bit adder chunks in terms of an input carry  $c_{in}$  and the  $p, g$  inputs (from the previous level  $P, G$  unit)  $p_0, g_0, p_1, g_1, p_2, g_2$ —provide all the logic equations and a logic gate schematic. **20**

(b) Using the above  $P, G$  and carry-generation units, and modified full adders (MFAs) that generate the  $p_i, g_i$  and  $s_i$  bits with inputs  $x_i, y_i$  (the bits to be added) and  $c_i$  (the carry in) as black boxes (properly labeled rectangles with i/ps and o/ps without showing any internal details), design a full CLA 12-bit adder. **40**

(c) Analyze the # of gate delays for the design of part[b]. **10**

2. Consider a 5-stage pipeline: IF, Dec&Read (ID), Exec (Ex), Data Access (MEM), Write Back (WB), each having a delay of  $1cc$ , as in the MIPS processor pipeline discussed in class (Chap. 6 of text and Lecture Notes # 14b). There are two possible architecture schemes for resolving/evaluating branches: (i) At the end of the Ex stage; (ii) At the end of the the ID stage (this can be done by inserting extra hardware that, for the BEQ instruction, for e.g., checks if the two read out registers are equal using XOR gates followed by OR'ing of the XOR outputs; read pp. 496-501 of the text.) In both schemes, the branch instruction is detected at the ID stage.

The *CPI* is defined as the # of cc's taken per instruction on the average, meaning that if  $n$  instructions take  $C$  cc's (to come out of the pipeline), then the CPI is  $C/n$ . Note that the CPI need not be an integer (e.g.,  $CPI = 1.2$ ) as it is an average.

(a) If 20% of the instructions are branches, then compute the CPIs for architecture schemes (i) & (ii) above, for the *NOP-filling* or *stalling* strategy to tackle branch or control hazards. **40**

(b) If 30% of the instructions are branches, then compute the CPIs for architecture schemes (i) & (ii) above, for the *fetch-next, flush-if-true* strategy to tackle branch or control hazards. Assume that a branch will evaluate to true 50% of the time and to false 50% of the time on the average. **50**

**Hint:** Figure out the average amount of **extra delay**  $d_{av}$  after a branch processing before which the correct instruction after a branch will exit the pipeline in each of the above 4 cases; note that  $d_{av}$  can be a real number as it is an average. Then allocate  $1 + d_{av}$  cc's (instead of 1 cc) as the time it takes for a branch to exit from the pipeline after the previous instruction exited from the pipeline (for

non-branch instructions it would take 1cc to exit after the previous instruction assuming no data or structural hazards). Proceeding this way, obtain the CPI.

3. Consider a 4 stage pipeline: fetch—IF (2cc), decode & read—ID (1cc), execute—Ex (4 cc for multiply, 7 cc for divide, 1 cc for all other arithmetic and logic operations) and write back—WB (1 cc). Assume that in a program *A* which takes 10,000 instruction executions to complete, there are 10% multiplies, 5% divides, and 85% of other types of instructions that take 1 cc in the execute stage. Assume that branches are detected in the ID stage and are evaluated in the WB stage.
- (a) How many cc's does it take to execute program *A* assuming for simplicity that there are no branches? **50**
- (b) How many cc's does it take to execute program *A* assuming that on the average there is one branch per 10 instructions (10% branches) and we use stalling (NOOP filling) for branches? **40**
- (c) How many cc's does it take to execute program *A* assuming that on the average there is one branch per 10 instructions (10% branches) and we use branch prediction? Assume that across all branches, True branches are taken 50% of the time and False branches are taken 50% of the time. Also, assume that branch prediction is correct 70% of the time and incorrect 30% of the time. **40**

**Hint:** If there are no multiplies, divides or branches, the CPI will be 2 cc's. Treat multiplies and divides as instructions requiring stalls of an extra 3cc's and 6cc's so that they exit the pipeline 5 and 8 cc after the previous instruction, respectively (just like we treated branches requiring extra delays in exiting the pipeline due to stalls). Also, an instruction ( $i + 1$ ) following a multiple cc ALU instruction ( $i$ ) taking  $x > 1$  cc's, will "make up" the 2 cc gap between them and reduce it to 1cc (the smallest gap possible), so that, if instruction ( $i + 1$ ) generally follows the previous instruction out of the pipeline after a gap of  $y$  cc's, ( $y = 2, 5, 8$  for regular/mult/div instr., resp.), then it will now follow the  $i$ 'th instruction after a gap of  $y - 1$  cc's. This reduction of 1 cc, can be "rewarded" to mul/div instructions, so that they can now be looked upon as having 4 and 7 cc's delay (after the prev. instr.) and all other instructions reverting back to a 2 cc delay—it is all a matter of which columns the credits and debits are allocated to, and as long as the total credits and debits are correct, the overall analysis will be correct. Finally, add in the extra delays for branch instructions in parts [b]-[c] as discussed earlier. After the CPI is computed in each case, you can obtain the total cc's for 10K instructions.