CS 520 - Fall 2017 - Ghose

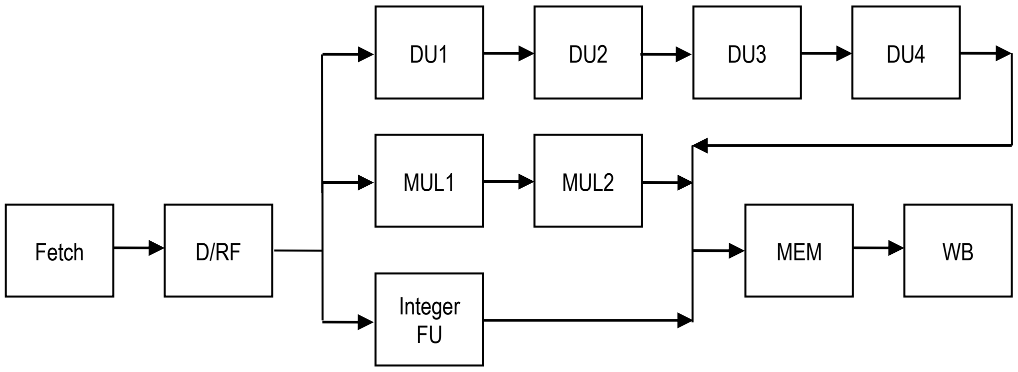
Programming Project 2: Extension of simulator for APEX developed in Project 1
THIS IS AN INDIVIDUAL ASSIGNMENT

DUE: Monday, November 6 midnight, NO EXTENSIONS

All demos to be completed by Friday, November 10.

For this programming assignment, you have to extend the simulator you have developed for Project 1 as follows:

EXTENSION 1: ***Incorporate an integer division unit (DU), pipelined into 4 stages*** (DU1 through DU4), each with a delay of one cycle. The addition of this function unit will lead to out-of-order completions and all relevant dependencies that may be violated by out-of-order completion need to be handled correctly. When multiple function units contend to use the MEM stage, the priorities for using the MEM stage are as follows: DU (highest), MULtiply FU, IntegerFU (lowest). The pipeline structure that results is as follows:



A new instruction accompanies the division unit, whose format and semantics are as follows:

Format: DIV <dest>, <src1>, <src2>, where all operands are registers.

Semantics: <dest> $\leftarrow$ integer part of <src1>/<src2>

EXTENSION 2: ***Add forwarding mechanisms from the last stages of the FUs to forward results (including flag values) to waiting instructions in the D/RF stage***. The forwarding happens at the end of the clock cycle when the instruction producing the results (including flag values) are in final stages of the respective FUs (integer, multiplier, divider). If the forwarded result(s) is/are the only source data that the instruction in D/RF was waiting for, it can be issued at the beginning of the clock cycle that immediately follows the one when forwarding took place. Remember that an instruction in D/RF can be issued only when ALL of its source operand are available, either from registers or through forwarding. Note that all three type of FUs can generate flag values and dependencies over flags have to be implemented properly. In a given cycle, forwarding from multiple sources can take place concurrently over independent forwarding paths.

EXTENSION 3: ***Add a jump-and-link instruction, JAL to implement a function call***. The format and semantics of the BAL instruction is as follows:

> Format: JAL <dest>, <src>, <#literal>, as in: BAL R3, R2, #-24
>
> Semantics: Saves the address of the instruction following JAL (PC-value of BAL + 4) in the register named in <dest> and transfers control to the address of the first instruction in the called function ("target address"), which is aligned at 4-Byte boundary. The target address is obtained by adding the contents of the register named in <src> with the literal. All instructions that have followed JAL into the pipeline are flushed. As in Project 1, the target address of JAL has to be converted to the address of a line in the text file that contains the code of the program whose execution is being simulated.

A return from this function call is simply implemented as a JUMP <dest>, <#literal>. where the literal value is zero, as in JUMP R3, #0 for the function called using JAL R3, R2, #-24.

What needs to be turned in: Zipped/tarred files of source, all design documents, test run outputs, as in Project 1. These need to be uploaded to Blackboard by midnight, November 6.