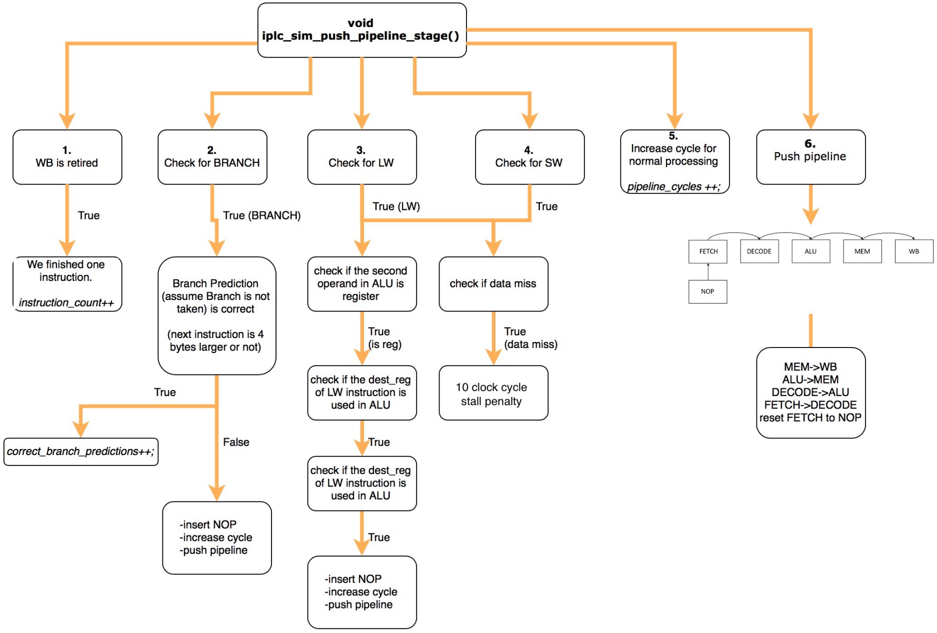
Pipeline Functions

The most complex function is: `void iplc_sim_push_pipeline_stage()`
The following picture is the flow chart for this function which shows how the logic works.

For the branch checking, because the branch decision cannot be made until the end of the EX stage and we need to know which instruction to fetch next, so we give branch a guess, assume it is taken or not taken. We assume the branch is not taken, if we are right ,then there is no problem and the pipeline keeps going at full speed; if we are wrong, then we need to insert NOP and push pipeline.

Then we need to check if there is any data miss in LW and SW, there is a 10 clock cycle stall penalty in the MEM stage for LW and WRITEBACK stage for SW.

Also if LW requires a register that is used by ALU, we need to insert NOP and push pipeline.


void iplc_sim_push_pipeline_stage()
1. WB is retired
True
We finished one instruction.
instruction_count++
2. Check for BRANCH
True (BRANCH)
Branch Prediction (assume Branch is not taken) is correct
(next instruction is 4 bytes larger or not)
True
correct_branch_predictions++;
False
-insert NOP
-increase cycle
-push pipeline
3. Check for LW
True (LW)
check if the second operand in ALU is register
True (is reg)
check if the dest_reg of LW instruction is used in ALU
True
check if the dest_reg of LW instruction is used in ALU
True
-insert NOP
-increase cycle
-push pipeline
4. Check for SW
True
check if data miss
True (data miss)
10 clock cycle stall penalty
5. Increase cycle for normal processing
pipeline_cycles ++;
6. Push pipeline
FETCH
DECODE
ALU
MEM
WB
NOP
MEM->WB
ALU->MEM
DECODE->ALU
FETCH->DECODE
reset FETCH to NOP

The rest of the functions are much easier. We just need to assign instruction with corresponding address, registers, and another instructions if it needs.