Checks in Issue Unit:

Starting with curr == 0 we loop and try to issue each instruction in the premem

```
while( numIssued < 2 and numInPreIssueBuff > 0 and curr < 4 ):  # curr is
current pre issue element
    issueMe = True
    # get an instruction from preissueBuff starting with element 0
    index = sim.preIssueBuff[ curr ]
    # make sure there is an instruction to execute
    if index == -1:
        break
```

Structural Hazard Check:

```
# see if there is room in the following buffers: structural hazard
# if instruction is a memory instruction and pre mem buff has both positions
with -1 in them
# don't issue the memmory instruction. Should be -1 in position 1 almost
always.
if sim.isMemOp( index ) and not -1 in sim.preMemBuff:
    issueMe = False
    break
 # if another instruction type same for preALUbuff
if not sim.isMemOp( index ) and not -1 in sim.preALUBuff:
    issueMe = False
    break
```

Process the instructions in the preIssue buffer in order from entry 0 to entry 3 and check that instruction for any hazard issues with other instructions in the preIssue buffer and with the instructions in the mem and alu pre buffers. If the instruction passes all tests, issue it to the appropriate mem or alu buffer.

We do RAW check with premembuffer:

The issue unit must make sure that before it issues from preissue buffer it does a RAW (read after write) check. This check makes sure that you do not issue an instruction that needs a register value that is still being determined by and instruction alreaday in flight or issue an instruction that WILL BE in conflict with an instruction earlier in order in the preissue buffer that will be writing to a register that this instruction needs.

PreIssue Buffer

0	ADD R2, R3, R10
1	ADD R3, R1, R2
2	SUB R4, R6, R5

3	AND R5, R1, R3

Lets say we have the above preissue buffer. We tried to process slot 0 but could't becuase there was in instruction in flight that was determining R10, so we try to issue slot 1. We look at the source registers for slot and compare them to instructions that according to the original code flow are ahead of them in other words, slot 0. Crud, we see that the R2 reg is a source in slot 1 but R2 needs to be determined by the slot 0 instruction before we can run it. So no issue. So we try to issue slot 2. None of the registers conflict with either of the destination registers of the two instructions that preceed it in the code flow and preissue buffer so we can now go on to the next check for this instruction.

The key is that if an instruction fails any hazard check it is delayed!

RAW check with preMem buffer:

If either source register of the instruction you are trying to issue (now slot 2) is the same as the destination register of either instruction in the preMem buffer then we have a hazard and don't issue and wait for the instruction in preMem to finish processing. This is like a LDUR problem and we will wait until it goes all the way through.

RAW check with preALU buffer:

If either source register of the instruction you are trying to issue is the same as the destination register of either instruction in the preALU buffer then we have a hazard and don't issue and wait for the instruction in preALU to finish processing. This is a data hazard.

RAW check with postMem and postALU

If either source register of the instruction you are trying to issue is the same as the destination register of either instruction in the postMem/postALU buffer then we have a hazard and don't issue and wait for the instruction in postMem/postALU to finish processing.

Do WAW check:

WAW check - later instruction tries to write an operand before earlier instruction writes it - pretty rare but possible in our simulator pipeline since mem and alu are stacked. Execution order important. If the destination of the instruction we are trying to issue is equal to the destination of the previous instruction we have a hazard. Check post buffs too.

If the instruction we are trying to issue passes ALL these tests then issue it and rearrange the preissue buffer and look at the same slot again.

```
if issueMe:
    numIssued += 1
    # copy the instruction to the appropriate buffer
    # the assumption here is that we will have a -1 in the right spot! Think we
will.
    if sim.isMemOp( index ):
        sim.preMemBuff[ sim.preMemBuff.index(-1) ] = index
    else:
        sim.preALUBuff[ sim.preALUBuff.index(-1) ] = index

    # move the instrs in the preissue buff down one level
    sim.preIssueBuff[0:curr] = sim.preIssueBuff[0:curr]
    sim.preIssueBuff[curr:3] = sim.preIssueBuff[curr+1:] # dropped 4, think
will go to end always
    sim.preIssueBuff[3] = -1
    numInPreIssueBuff -= 1
```

ELSE: look at the next slot!