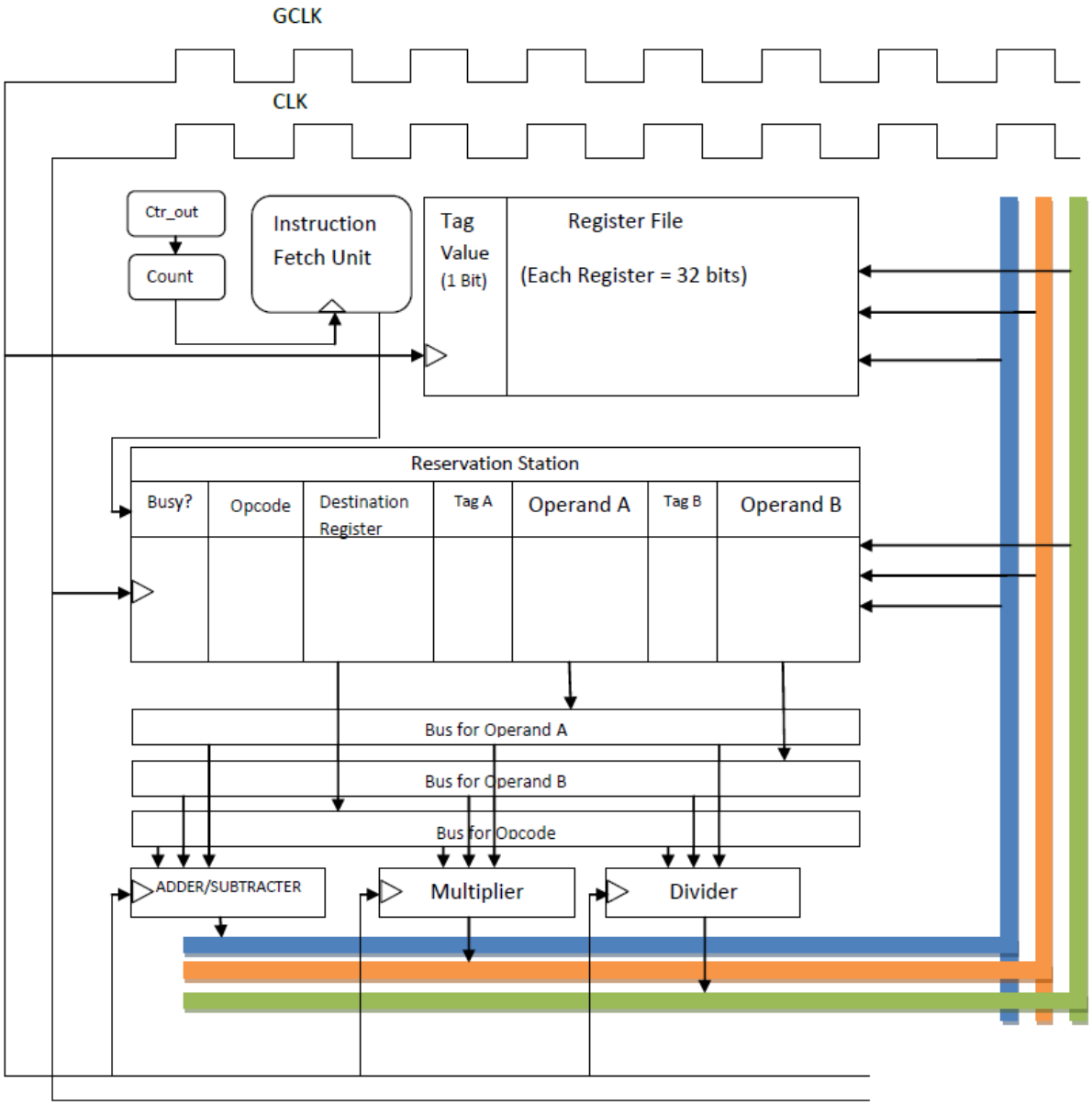
Scheduling Using TOMASULO'S Algorithm

In this assignment, a dynamic scheduling process of instructions is implemented based on a variation of Tomasulo's algorithm. The implementation is done in verilog (Modelsim v6.6 student edition). The basic functional block diagram of the implementation is as shown below:



The Flow is as follow:

The instruction flow in Tomasulo algorithm is shown in Figure 1. Each instruction on arrival fetches its two operands from the register file that has 16 registers. Each register in this file holds either an actual value, or a tag id indicating the reservation station that will produce the register value when it completes. The instruction and its operands (either values or tag ids) are stored in a reservation station. The reservation station watches the results returning from the execution units, and when the tag of a result matches one of its input operands, it records the value in the place of the tag. When the reservation station has both values of its operands, it may issue its instruction to an execution unit if it is free. When the tagged result

returns from the pipeline, the reservation station is cleared, and the result value is stored in the destination register. There is also a stall output indicating that an instruction cannot currently be received. A stall can happen when all the reservation stations are full. In the whole system, the instructions are read from during the positive edge, and written to during the negative edge of the clock cycle.

Register File: It consists of 16 Registers. Each register is of size 32 bits. Each register file has a "Tag Value" associated with it. Tag Value = 1 means that the register contains the TAG (Reservation Station ID) of the operation that it is waiting to complete to get its own value. The register file is compared for tag at the end of any computation (at a negative edge of CLK).

Reservation Station:

The reservation station receives two operands (or their tags), opcode and the destination register ID for the operation. In addition, each RS consists of a "busy bit" to indicate if the operands in the corresponding RS are busy in the functional unit or not. *The RS is written to at the negative edge of the CLK*. When all reservation stations are busy, the fetch system is brought to a stall *by pulling down CTR_EN to zero*. When at least one reservation station becomes free, the stall is ended. The RS is written into at the negative edge of CLK from the Instruction Fetch Unit as well as during functional unit write back.

Execution Units:

There are three separate execution units namely Add/Sub, Multiply and Divide. The Latencies for each of these executions was introduced using artificial delays in verilog. The operands are sent into temporary variables (A and B) of each reservation station in the positive edge of CLK. At a time, only one instruction from the RS is sent to the functional unit. The selection logic used is location-based selection logic. Once execution is completed, the result is written back to the corresponding TAG entry in the register file and the reservation stations in the negative clock cycle of CLK. Each functional unit has its independent CDB so that means multiple write backs can occur at a negative edge.

At Halt, all the reservation stations and functional units are cleared to XXXX (don't care).

For the given instruction stream, the final state of all registers is shown below:

Register Number	Tag Value	Register Content
0	0	10
1	0	0
2	0	5
3	0	5
4	0	25
5	0	0
6	0	0
7	0	5
8	0	5
9	0	5
10	0	5
11	0	5
12	0	5
13	0	5
14	0	5
15	0	5