DESIGN DOCUMENTATION

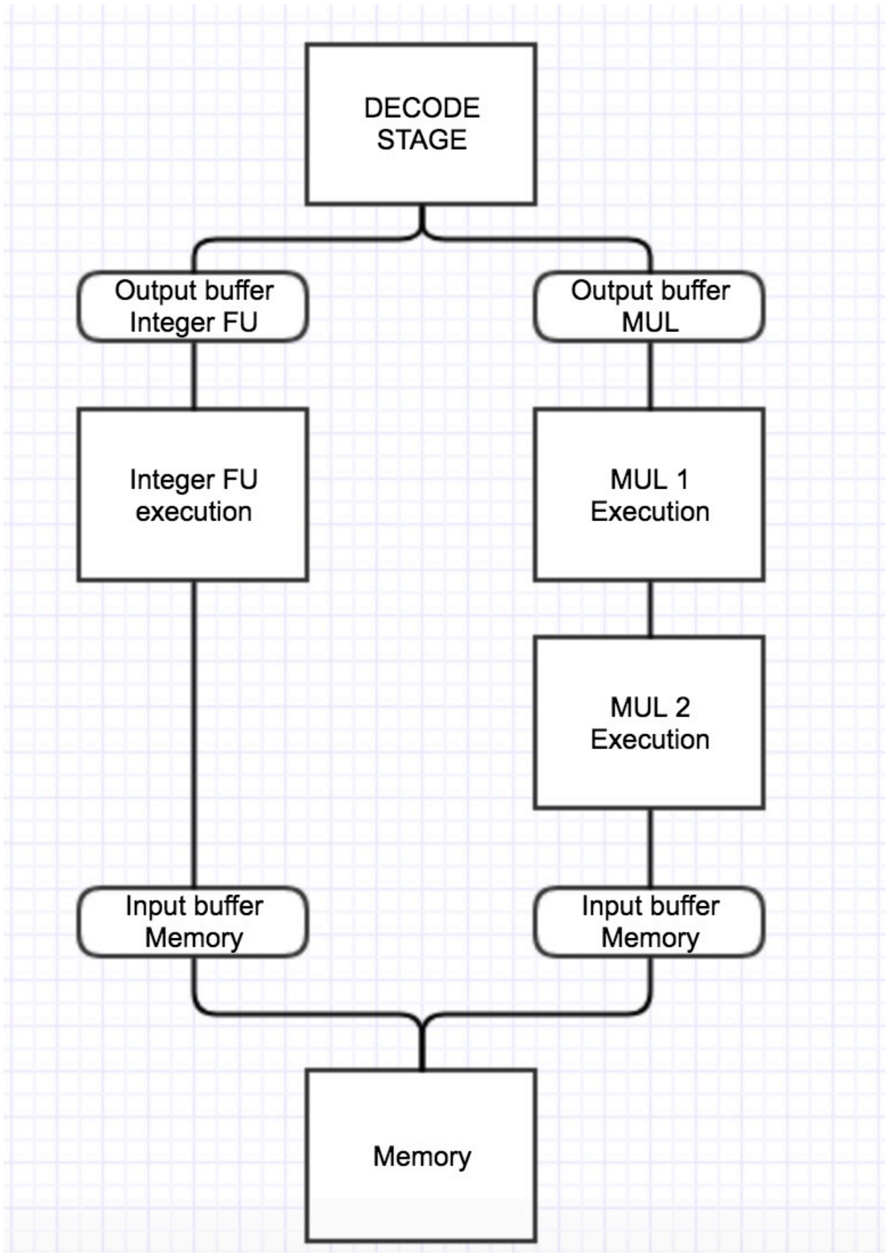
STRUCTURE's USED:

1. struct code_line : I've used this structure to store a single instruction from the input file with the line number and the address.
2. struct code_memory : Code memory is used to create an array of structure code line, which helps to store all the instructions together in the array.
3. struct data_memory : Data memory holds the memory. It contains the address and the array of memory locations. The total size of the array to store the memory data is 4000.
4. struct registers : It is a structure to store a single register with the value, address and the status of that register.
5. struct register_file : It is a register file which has the structure registers as its attribute. This structure is used to store all the register together by creating a struct array of registers structure. The total number of registers are 16.
6. struct stats : This structure is used to store the total number of cycles.
7. struct instruction_info : I've used the structure instruction info to store the all the information about the status of each stage. I've used this structure to to initialize buffers so that the information can be passed to the next stage.
8. struct stage : This structure is used to store the input buffer and the output buffer all the stages in the pipeline.
9. struct flags : I've used the structure to store all the flags. It contains a zero flag, jump,BZ,HALT and BNZ. Through these flags the the branching instructions are executed.

ALGORITHM:

1. In the main function first the values are initialized to zero to empty the structures and global variables to avoid garbage values.
2. The file is read to store all the instructions with the addresses and file number in the structure code_line.
3. A while loop is used to implement the pipeline in loop, with the termination condition entered in the command line.
4. There are in total 5 major stages with a break up of the execution stage in two different flow, i.e. the arithmetic_execution and the multiplication_execution.
5. All the stages are called in the reverse order, this is because the buffer is filled out and is given to the next stage in pipeline, so that the next stage pulls the data from the previous stage. So to avoid execution in sequential order the stages are called in reverse order.
6. All the stages has 2 buffers the input and output buffers. The input buffer is used to fill all the data in from the previous stage and the output buffer is used to give the data to the next stage.
7. The execution is divided into 2 flows. And to ensure that every time they are in parallel, an additional buffer in is introduced in decode stage and memory stage.
8. For the instructions going in the arithmetic execution stage(Integer FU), the output buffer A is populated in the decode stage and the instruction that is to be executed in Multiplication stage the output buffer B of decode is populated.
9. The same is implemented in Memory stage. The data flowing from multiplication stage is are populated in the input buffer multiplication and the data flowing from the Integer FU are populated in input buffer execution of Memory.
10. All the instructions flows from the memory and the write back stage.

11. For register-to-register instructions the registers are written in the register file in write back stage, and for the instructions that have memory operations perform memory read or write in memory stage.
12. Once the HALT instruction is triggered in the decode stage, the execution stops in and the instructions prior to the HALT stages are flushed from the pipeline, and the execution is stopped.
13. Following is the detailed diagram for the break up of the execution stage.



The following things are not implemented in the code:

1. The branch instructions are handled, however due to some bugs the branch instructions are not executing.