**Mohamed A. Mohamed 900182267**

**Nada El-Shenawy 900182782**

**Project Milestone 3 Report**

**Design Approach:**

In this milestone, the data path has become pipelined with a restriction of five stages for each instruction: **IF, ID, EX, MEM, and WB.** Each stage of each instruction is executed with a clock cycle "Not completely right, as the slow clock is used for multiple reasons mentioned later". The reason for changing the data path to be pipelined is the chance of increasing the instruction throughput by executing different instructions simultaneously. As required, our program has a single ported memory where you can find the instruction memory and data memory. Here comes the problem of Clock Cycle. To avoid Structural Hazard of requesting an access for fetching instructions and loading from a memory, we have implemented the slow cycle each slow cycle is two cycles of the normal clock. In that sense, we need to have the memory address that will be chosen depending on the slow clock value. Then, it will be easy to inform the memory whether I will consider the memory as instruction memory or data memory. As a bonus Feature, deciding to branch or not is known in the decoding stage. This has been executed by taking into consideration the forwarding and stalling to avoid data hazard. There are two forwarding units: one for load-use data hazard and the other one for branching. The normal forwarding unit only forwards the value from memory/write back stage to execution stage if needed since our design follows the slow clock.

if ( MEM\_WB\_RegWrite && (MEM\_WB\_RegisterRd != 0) && (MEM\_WB\_RegisterRd == ID\_EX\_RegisterRs(1or2)) )

forward(A or B) = 1;

else forward(A orB) = 0;

The second forwarding unit, branching, forwards the value from execution stage to decoding stage instead of waiting to reach the write back stage.

if(IF\_ID\_OPCODE == `OPCODE\_Branch && EX\_MEM\_RegWrite && (EX\_MEM\_RegisterRd != 0) && (EX\_MEM\_RegisterRd == IF\_ID\_RegisterRs(1 or 2)))

forward(A or B) = 1;

else

forward(A or B) = 0;

Also, we need to stall when there is a need to check from a register where it loads from a memory and then branching.

if(EX\_MEM\_OPCODE == `OPCODE\_Load && IF\_ID\_OPCODE == `OPCODE\_Branch && EX\_MEM\_MemRead && (EX\_MEM\_RegisterRd!=0) &&(EX\_MEM\_RegisterRd == IF\_ID\_RegisterRs1 || EX\_MEM\_RegisterRd == IF\_ID\_RegisterRs2))

stall = 1;

else

stall = 0;

The bonus features automatically solved the problem of flushing, as we predict in the decoding stage. Finally, Ecall instruction is used to terminate the program just by halting the PC even if there are further instructions needed to be implemented.