# Report

## Thursday (6/01)

We started our project keeping in mind the exact sequence of fetch cycle. Our 1st priority was to create the PC. We found this difficult to implement since we couldn’t automatically increment the PC even after trying a few methods and updating the PC manually was wrong in terms of efficiency and proper working of our processor. By end of the day we figured out how to create an increment-er using an adder, a constant and a register. We used a MUX in order to make a choice between initial storing of all operations and fetching each operation at a time as we progress.

## Friday(7/01)

We built our MAR using a register to store fetched instruction while the PC moves to the next instruction. This instruction is broken down using a splitter into 3 components. The first 4 bits were used as OPCODE, next 3 bits as register address and 9 bits as memory location. In this case we made a mistake with splitter connections as we were not clear about the highest and lowest bit.

We had an issue while making connections to the memory as we had to make one connection where we enter the initial data into memory manually and another where we give entire control of memory to the processor to read and write. We used a MUX which shared its select line with MAR so that fetch and execution work in sync

## Saturday (8/01)

Today we built the register file. We connected the clock input to the same clock of fetch and execute in order for the data to get stored and accessed from the registered in 1 clock cycle.

## Sunday(9/01)

Today we decided to build the ALU system we had two challenges. We had to bring one operand from the register and the other operand from memory. When we used decoder it did not work as data was not passing through. After trying to used registers for temporary storage we realized that this would complicate the system and increase clock cycles. After research we decided that a DMUX would allow us to pass data depending on the input from select line. Hence ,used one for register operand and memory operand.

## Monday(10/01)

After we got the result from operations we did not know how to store output back into the same register from which we got our operand. We tried to use a MUX for this function with opcode select line but we faced an issue with this approach. Later we realised that in the register file only 1 register was being selected for an operation so even if we connected the output directly to the input line of register it would automatically store our result in the same register.

## Tuesday(11/01)

There was an issue with the memory input for the shifter and rotator. We realized that to get valid output for a 16 bits of data we need to rotate or shift it by maximum 15 times. This can be achieve by the starting 4bits and hence a splitter was used to choose 4 bits.

## Wednesday(12/01)

The processor is always reading from memory while performing instructions, it writes only when we have manual control and at the time of STORE and XCH Instructions. Hence we created mechanisms for their respective opcodes.