DIPEN KUMAR (2018CS50098)
UMESH PARMAR (2018CS50424)
COL216 ASSIGNMENT-6

This Assignment was the extension of previous assignment on Mips processor. In this assignment we have to add instructions on conditional and unconditional jumps. We also implemented stack memory which grows downwards. We have designed our test case which is a recursive program to show procedure that can call other procedures(in this case two) using stack.

In this assignment along with the above mentioned task, we tried to improve our processor design. In this assignment we can up with an optimized an efficient processor design which takes 1-cycle to perform- add, sub, sll, srl, jr, j, jal, beq, bne, blez, bgtz and 2-cycle to perform- lw, sw when used pipelining. We achieved this by concurrently assigning the value of address and saved 1-cycle which was used for address to update its value.

Also we improved on individual process-time of each instruction. Earlier our implementation has many stages for each instructions. But now we have only 2-stage in add, sub, sll, srl, jr, j, jal, beq, bne, blez, bgtz and 3-stage in lw, sw.

We have four state in our implementation i.e. {0,1,2,3}

1. State 0- We start our processor from state=0 where it concurrently assign the value of the address of the instruction to be processed in next cycle. In short we are processing the first stage of the instruction to be executed next cycle.
2. State 1- Here we do two things i.e. we are executing the instruction whose address was concurrently assigned the previous cycle and also assigning the address of the instruction to be processed in next cycle. Technically speaking we are processing the second stage of the instruction which is currently being executed and also processing the first stage of the instruction which will be executed next cycle.
3. State 2- This state is for lw and sw instructions only because they have three processing-stage in pipeline. In this state third process-stage of lw and sw is processed. If we find the instruction to be any one of lw and sw while executing it in state-1 in their second process-stage, we shift our state from 1 to 2 since we have to wait for next cycle to get the value at required address in case of lw and write the instructed value at required address in case of sw. Next cycle i.e. in state 2 we have the required value and hence complete our lw and sw execution finally (stage3). Also here we concurrently assign the value of the address of the instruction to be processed in next cycle i.e. we are processing the first stage of the instruction to be executed next cycle.
4. State 3- This is idle state (doing nothing since all instructions has been executed)

The output instructions remain the same as for Assignment 4. We show number of cycles taken to execute entire instructions on seven-segment display in hexadecimal format when switch is ‘1’. In other case when switch is ‘0’ we displayed the lower 16 bits of the output register in the last instruction on seven-segment display in hexadecimal format.