Introduction:

The objective of the assignment is to simulate the datapath of a 32 bit mips architecture responsible for executing the following list of instructions in python.
Add, Sub, logical And, logical Or, set less than, add unsigned, add immediate unsigned, subtract unsigned, load word, store word, load upper immediate , break, branch if equal and branch if not equal, unconditional jump.

Design & Implementation:

The fundamental step is to understand the CPUelement described in precode as it forms the basis for all the other components involved in the architecture. Once every individual element of the pre-defined architecture (source: computer architecture and organization, David Patterson) is implemented successfully by checking for test cases, they subsequently can be connected by a mips simulator class. The final step is to run the simulator on given .mem files to ascertain the veracity of the datapath.

Now that the basic flow of control is explained, the set of the elements constituting the architecture is enlisted below:

- constant element responsible for increasing the program counter by 4 bytes.
- an adder for performing the above operation along with another adder for calculating the branch address in case of a branch type instruction.
- an instruction memory for fetching the instruction which in itself is a 32 bit binary code.
- a control unit that decides which operation and how should it be performed (“how” here refers to choice of registers and control signals to multiplexers and the other elements). it has two input signals one of which is opcode of the instruction. In case of Rype instructions the 6 lowest bits of the instruction in conjunction with opcode decides the output control signals. One of the 11 control signals is a 4 bit output while rest of them are a bit.
- a register file which as the name suggests is used for storing data and addresses in its collection of 32 registers. It reads three register numbers present in the instruction one of which is a destination register while always sending out the values present in the register. The fourth input to the data to be written onto the destination register in case of a load or load upper immediate operation.
- an element for shifting input by 16 bits and an element for shifting input by 2 bits.
- an alternate and element that takes in 3 control signals from the control unit as inputs and sends back a single control signal used by the corresponding multiplexer to decide whether or not to use the branch address or a regular incremented pc.
- an alternate shift element used to take in two input signals, one from the instruction memory and an incremented pc to produce a concatenated unconditional jump address. This address is sent as one of the input signals to the respective multiplexer. the input from instruction memory is shifted left by 2 bits while only the 4 most significant bits is extracted from the other input. Concatenation can be implemented either by a simple addition or a logical OR operation.
- a data memory to store data items at valid memory addresses. Its purpose is to assist load and store operations depending on the values of its two control signals. If it is a read operation, the incoming address in read port is read and corresponding data item is sent as an output while in case of a write or store operation register file output is written to the address calculated by alu unit.

- an arithmetic logic unit to perform addition, subtraction, logical And and logical Or. In case of instruction that involves summing an offset, It takes two input signals one of which could be a sign extend 32 bit offset, while the other is a base address held by register. Otherwise it manipulates two registers and returns a value based on four bit control signal.

- 6 multiplexers each with a functionality depending on what element is in its proximity

Unittest is used to test independently if the components function as per our expectations. Almost every type of instruction or at least one of each kind (branch, memory and r-type) is tested to ensure Alu sends out the desired output signals.

Results and Discussion:

self.elements = [self.constant4, self.adderpc, self.instructionmemory, self.controlunit, self.altershift, self.mux_writereg, self.registerfile, self.shift16, self.signextend, self.shift2, self.addershift, self.mux_regoutput, self.alu, self.alterand, self.mux_branch, self.mux_jump, self.datamemory, self.mux_datamem, self.mux_shift16, self.registerfile]

A crucial point to keep in mind is to preserve the order of elements placed in the list above for valid control flow as It would not make sense to call the data memory without knowing what the control output signals are.

Or to put it another way the following flow should be maintained: instruction fetch, instruction decode, alu operation, memory operation, write back stage. These are also the 5 stages of a data pipeline.

Another pitfall to take care of ensuring the overflow is ignored when the data is converted from signed to unsigned in add unsigned operation as the resulting output is susceptible to be more than 32 bits if both the operands are large. Thus the result should also be converted to an unsigned equivalent before its use.

Problems can be encountered trying to access a data item at memory location which does not have any value. Hence its beneficial to use the get() function of memory which returns 0 if nothing is present in the address.

One has to be cautious of the order of the arguments sent to multiplexers if we are to expect the correct output signals.

Result of a fibonacci series is as expected and can be verified by checking if the 65th number in the series is present in one of the registers.

Selection sort successfully sorts the given 13 numbers and can be checked by uncommenting the respective lines in the simulator file.

Conclusion:

The single cycle implementation of 32 bits mips architecture produces expected resulted when ran on all the available .mem files, thereby enabling the constructed data-path to execute a predefined list of instructions.