Project C Report

We made the processor with pipeline. We inserted four pipelines. We demoed with the bubble sort program. We included nop to avoid all the data hazards and changed the branch addresses by 2 also. The Mips single cycle processor performs the tasks of instruction fetch, instruction decode, execution, memory access and write back all in one clock cycle. The first PC value is used as an address to index the instruction memory which supplies a 32 bit value of the next instruction to be executed. The instruction is then divided. The opcode of the instruction decides which type of instruction is executed. The type of instruction then determines which control signal is to be asserted and what function the ALU is to perform (this is the decoding of instruction basically). In reg file we have two register reads and one write in one clock cycle. The reg file reads in the requested addresses and outputs the data values contained in those registers. These data values are operated on by the alu whose operation is determined by the control unit. In the final step we write the alu result back to the reg.

The instruction fetch unit is to obtain an instruction from the instruction memory using the current value of the PC and increment the PC value for the next instruction. It has a 8 bit program counter register, an adder to increment the PC by four, the instruction memory, a multiplexor and a and gate used to select the value of the next PC.

The next is the instruction decode unit. Its main function is to use the 32 bit instruction given from the instruction fetch to index the regfile and obtain the register data values.

The control unit examines the instruction opcode bits and decodes the instruction to generate nine control signals to be used. The register destination control signal determines which register is written to the register file. The Jump control signal selects the jump address to be sent to the PC. The branch control signal is used to select the branch address to be sent to the PC. The mem read control signal is asserted during a load instruction when the data memory is read to a load a register with its memory contents. The memtoreg control signal determines if the alu result or the data momory output is written to the register file. The ALUop control signals determine the function that alu performs. The memwrite control signal is asserted when during a store instruction when a registers value is stored in a data memory. The alusrc control signal determines if the alu second operands comes from the register file or the sign extender. The regwrite control signal is asserted when the regfile needs to be written.

The exmem unit contains the alu which performs the operation determined by aluop signals. The branch address is calculated by pcplusfour.

The write back is used for load and store. The load instruction asserts the memread signal and uses the alu result value as an address to index the data memory. The read output data is then subsequently written into the regfile. A store instruction asserts the memwrite signal and writes the data value previously read from a register into the computed memory address.

Pipelining the MIPS processor introduces events called hazards, which prevent the next instruction in the instruction stream from being executing during its designated clock cycle. The types of possible hazards include structural, data and control hazards. Data hazards arise when an instruction depends on the result of a previous instruction in a way that is exposed by the overlapping of the instructions in the pipeline, thus causing the pipeline to stall until the results are made available. One solution to this type of data hazard is called forwarding, which supplies the resulting operand to the dependant instruction as soon it has been computed. While forwarding is an exceptional solution to data hazards it does not resolve all of them. One instance is when an instruction attempts to read a register value that is going to be supplied by a previous load instruction that writes the same register, called a load-use hazard.

At the same time the load instruction is reading data from memory, the subsequent instruction executing in the execution stage with the wrong data value. The only solution here is to stall the pipeline and wait for the correct data value being used as an operand. In order to detect such hazards, MIPS introduces a Hazard Detection Unit during the instruction decode stage so that it can stall the pipeline between a load instruction and the immediate instruction attempting to use the same register.

Control hazard are also known as a branch hazard. These hazards occur when there is a need to make a decision based on the results of one instruction while other instructions continue executing. While the register values are compared, other instructions continue to be fetched and decoded. If the branch is taken, the wrong instructions are fetched into the pipeline and must somehow be discarded. A common solution to these hazards is to continue instruction execution as if the branch is not taken. If it is later determined that the branch is taken, the instructions that were fetched and decoded must be discarded which can be achieved by flushing some of the pipeline registers. Flushing means that all values stored in the pipeline registers are discarded or reset. However in order to reduce the branch hazard to 1 clock cycle, the branch decision is moved from the memory pipeline stage to the instruction decode stage. By simply comparing the registers fetch it can be determined if a branch is to be taken or not. The pipeline bubbles will be 3 cycles and 2 cycles, respectively, for each taken branch. If a branch is resolved at the EX stage, instructions at the IF and ID stages will be flushed. If a branch is resolved at the MEM stage, instructions at the IF, ID and EX stages will be flushed.