Connor Greenwood
Farhan Khan
EC551 Lab1 report

Overall, we implemented a 3 stage pipelined cpu. Initially we had planned on using a 5 stage pipeline, however this was too large for synthesis, so we had to scale back and switch to a 3 stage pipeline. The three stages of our pipeline were:

1. Instruction fetch
    a. Got instructions from memory
    b. Increment program counter
    c. Mux between jump address, reset address, and program counter address
    d. Write new PC to register file
2. Instruction Decode
    a. Decode instruction into opcode and registers
    b. Send opCode data to control unit for further decoding
    c. Get register data
    d. Handle halting
    e. Handle data forwarding
3. Execute/Memory/WriteBack
    a. Execute Alu instructions
    b. Handle memory
    c. Write data back to register file

The bulk of the work is done in Execute/Memory/WriteBack

Initially, we planned on doing a 5 stage pipelined cpu because it allows the execution sections to take around the same amount of time, splits up memory and execution, and allows the write back data to be held constant for an entire cycle.

Doing a 3 stage pipeline is a compromise on size and maximum speed. This is an ok compromise to make, because it is unlikely the FPGAs are fast enough to make a difference.

Therefore, the decreased number of registers is desirable for this FPGA, as the 5 stage pipeline was unable to synthesize due to size limitations.

INSTRUCTION FETCH

To implement the instruction fetch stage of the pipeline, the cpu reads the program counter in from the PC reg. Then, it muxes this with 16’d31, to reset the program counter if the signal to reset the program counter is sent. After this, it gets muxed again with the jump target, if the instruction is a jump and the conditions for jumping are met. After this, this PC is sent into memory as its own port to read in the program data from memory. This gets read, then put into a pipeline stage register to wait for transfer to the Instruction Decode stage. The PC is also incremented by one, and this value is stored in a register to wait to become the next PC for the next instruction.

INSTRUCTION DECODE

Most of the control for the CPU is done in instruction decode stage. The instruction is decoded into opCode, arg1, and arg2. opCode is then passed into the control unit, which determines which control flags to set, such as halt, memWrite, memRead, aluOperation, etc. Arg1 and Arg2 are sent into the register file, and their outputs are muxed with a forwarding unit for the data currently being written into the register file. This is required to avoid RAW hazards, so there is a forwarding control unit that handles this, and sets flags to forward the current writeBack data. The decision to forward is done by checking if the current register being read is equal to the current register being written to, along with the regWrite flag is set in the EX stage.

The next thing the Decode stage does is handle jump logic. It creates the jump address by concatenating arg1 and arg2. It then decides the jump logic by

- If unconditional jump, just jumping
- If conditional jump, it checks the equal register in the EX stage.

Decode phase also handles the halting. When it receives a halt signal, it sends a signal to stall the entire pipeline. Every register in the pipeline stops reading. Then it waits until the positive edge of the global resume signal. This is sent in through the top module.

EXECUTION PHASE

The ALU is simple, and just uses a basic case statement to choose between which operation to output. There is a mux that goes into the ALU which zero extends an immediate value, and chooses that as input two for the ALU if the control unit in decode determined this was an immediate instruction.

The memory unit takes in multiple inputs and outputs. It takes in the address, data memory, and program counter as inputs, and output the instruction read by the program counter and the memory at address. If the writeMem flag was set by control, then memory writes the data into the memory address specified by address. We also implemented the mov [Rn], [Rm] operation. This is handled by, within the memory, moving the data at address dataIn to the address at address. This allowed for minimal changes elsewhere, as all that was needed was the addition of a new control flag.

The writeback phase is the most simple. It simply muxes the output from memory with the output from the ALU, and writes that data back into the register file if the writeBack flag is set.