Gerald Brennan (gbrenna1@binghamton.edu)
Stefan Bossbaly (sbossba1@binghamton.edu)

1. Pipeline

The pipeline consists of multiple functions that represent the implementation of each individual section of the pipeline desired. This is accomplished in a “copy then execute” manner traversing the pipeline in reverse order. This is demonstrated by the following flow diagram:

As shown above “copy then execute” means that each stage specific entry is updated with the previous stage's day on the current frame, then executes the stage's specific logic depending on the instruction information passed on from the previous entry. For instance, in the beginning of the function the wb entry is updated with the entry the mem stage executed on during the previous stage. Then the wb stage is executed with the newly transferred instruction details. This occurs for every stage of the pipeline in reverse order.

Special cases are calculated for what info is transferred and whether an instruction goes to the next stage based on Dependency and Forwarding information calculated during execution. There are two stages where forwarding takes place during the main execution loop. The first set of forwarding covers the special Load/Store case that will be described later. It happens after the 2nd alu stage and delay stage have been updated. It forwards information required by the store as it enters the mem stage in order to execute. The 2nd batch of forwarding contains the main forwarding methods. These methods occur after the 2nd stage of both FUs have completed execution and forward whatever data is needed to the dr/f stage.

Before the dr/f dispatches any instructions into their respective FUs, a blocking function is utilized to

determine if any flow dependent functions have any source registers that have not yet been released by previous instructions. This method checks status bits of individual instructions to make sure their source registers are either freed or forwarding can get the result to the requesting entry in time for it's execution to occur. Once the blocking function is satisfied, the instructing is filtered into either one of the FUs based on a boolean computation choosing whether the instruction is an ALU instruction or a branch instruction. After the instructions are filtered the dr/f and fetch entires are updated with new instructions.

2. Dependencies

Since instructions are dispatched and executed in order, the only dependencies that need to be checked/acted upon are flow dependencies. The flow dependencies are calculated in the decode stage, setting the source valid bits in the decode entry based on the availability of the source registers required to run the newly decoded instructions. Until the dependencies are resolved (either through forwarding or the dependent instruction finishes executing) the instruction will be held in the dr/f stage. Every frame while the instruction is being held in the d/rf stage, the valid bits of each source register is checked. At any point if both of the valid bits show the source registers are no longer dependent on executing functions the instruction will be sent to it's respective FU and the next fetched instruction will begin the dependency check for the following frame.

There is a special flow dependent case in which a STORE instruction is using a register needs information calculated by an immediately preceding LOAD instruction. This dependency isn't realized until the STORE instruction tried to enter the mem phase. This special case lets the STORE instruction leave the dr/f stage even if the register isn't valid yet. This dependency will be cleared in the forwarding stages.

3. Forwarding

Forwarding is implemented in an attempt to reduce the amount of flow dependencies that lag the pipeline during execution. Since the ALU is a 2 stage FU, not all flow dependencies can be fixed with the use of forwarding but there are many cases in which forwarding does help the system. As shown in the flow diagram in section 1, most forwarding occurs after the FUs have both completed execution and contain information that can be used in the current instruction occupying the d/rf entry (again since this is copy then execute in reverse order, the current dr/f instruction doesn't have to wait until a steady state is achieved before receiving forwarded data). The dr/f entry can receive forwarded data from the mem stage, 2nd ALU stage, the delay stage, or the branch stage. This is done by checking to see if any source register required by the dr/f stage instruction is freed up after execution on the last frame. When this is freed, the validity of the source register is updated. Once all source registers are validated the instruction and proceed into an FU for execution.

A special forwarding case is also needed for the LOAD/STORE flow dependency discussed earlier. After wb is executed and the STORE is about to enter the mem stage, the LOAD instruction has a result for it's destination register but still needs to write the data to the register. This result is forwarded to the STORE instruction that is entering mem to validate the source register dependent on the load instruction.

4. Interface

The interface is the user input that can simulate the processor created. It requires a single input in a text file titled 'list.txt' be copied to the root folder of the project containing the list of instructions required to be run by the processor. After the list is populated the program can execute. It recognizes 3 commands: Initialize, Simulate <#>, and Display. Initialize is needed to be run before every instance. It

cleans out any clutter left behind from past simulations and populates the internal instruction dispatcher with instructions from the input list. Once initialize is run, Simulate followed by a number will execute that many cycles of the program, or until a 'HALT' instruction is executed. After execution a Display command can be issued to print out the contents of memory and registers to the console.