Ayush Tibrewal & Anton Melnychuk CPSC 420

Lab 2: Pipelined PARCv2 Processor

Abstract

In this lab, we extend a pipelined PARCv1 processor into a full-featured PARCv2 design by implementing key performance enhancements. The primary objectives include upgrading the control unit to support the complete PARCv2 instruction set, incorporating bypassing logic to mitigate pipeline stalls, and integrating a pipelined multiplier/divider (MulDiv) unit. These enhancements collectively enable the processor to execute a broader range of instructions with improved efficiency by reducing cycle counts and minimizing the performance penalties.

Design

Objective 1

Objective 1 was fairly straightforward. After using the `parc-isa.txt` file to read through and understand the PARCv2 commands, we added these commands into the core control signals, along with the appropriate signals carried by each instruction. We also added the branching logic that commands such as `beq`, `blez`, `bgez`, etc. need. This allowed us to have a functional processor, albeit with a very conservative stalling logic that ensures we avoid all data hazards until the previous instructions are complete.

Objective 2

The processor's core logic remains largely unchanged when bypassing is added, but we change the stalling logic in the Decode stage. Rather than stalling whenever an instruction at a later stage will be written to a register, we introduced a mux which allows for computed values to replace the register being used by the instruction we are decoding. Since the value to be written by an instruction such as `add` is known by the end of the eXecute stage, there is no

reason to stall for 2 additional clock cycles before allowing that value to be used. As demonstrated in Figure 1, this mux is used to modify the value read from the register file in the case that a future instruction plans to write a value to the register being read.

We also have prioritization for these bypassed signals. Recalling that the instructions in earlier stages are later in the execution, we prefer to bypass from the eXecute stage over the Memory stage, and the Memory stage over the Writeback stage. This means that if a register will be written to by multiple instructions further in the pipeline, the value from the latest instruction (in execution order) will be used.

Unfortunately, not all stalls can be removed with this bypassing logic. Of course, stalling must occur when a unit such as muldiv, imem, or dmem takes multiple cycles, since we must wait for the result, which "backs up" the previous stages. Further, we also have load-use hazards. If the instruction in the eXecute stage is a load, then the value is not actually known during this stage since the dmem unit returns a result during the Memory stage. During Objective 3, these issues will become even more prominent.

Objective 3

Finally, in Objective 3, we changed the muldiv unit to an imagined pipelined version. To do this, we needed to introduce two additional stages, PostMemory and PreWriteback. In this implementation, we send the muldiv unit a request directly from the Decode stage. Then, after the 3 stage pipeline internal to the muldiv unit, we get a response during the PreWriteback stage, which combines with the PostMemory result in a mux. We essentially duplicated most of the bypassing logic. However, similar to how loads in the eXecute stage require stalling and cannot be bypassed, muldiv operations in the eXecute, Memory, and PostMemory stages cannot be bypassed since they have not yet been computed. As expected, this means the Decode stage also

needs to stall if it needs to use the result from a muldiv operation that has not yet reached the PreWriteback stage.

The additional complexities in stalling, bypassing, and the natural consequences of adding two stages meant that we ran into a couple of bugs as we tried to implement this objective. The biggest one is discussed below in the Testing Methodology section. Besides this bug, we generally had to be very careful about which wires were used for bypassing, especially since some of the muxes such as the muldiv selection and the dmem selection moved around, either moving stages or moving relative to other components of the processor.

Testing Methodology

For testing, we relied primarily on the provided assembly tests that focus on each of the operations in the PARCv2 ISA. Since we ran into some issues with the implementation of the muldiv unit, we chose to focus on this for our custom test, `parcv2-custom-test.S`. The primary test was when this is run in the random sim, where the muldiv unit is likely to need stalling due to a load in progress. Namely, a bug we had in our Objective 3 implementation, where the stalling for the muldiv unit was not correctly implemented. This had not been seen from the usual mul test since it is only an issue when multiple instructions use the muldiv unit in a row. This also let us test the bypassing logic around the muldiv unit.

Evaluation

The bypassing processor does quite a bit better than the stalling processor in examples such as bin search where most of the operations rely on registers that are written to in a previous instruction. On the other hand, as one would expect, the much faster muldiv unit (3 cycles vs the 33 cycle iterative implementation) means that the final processor is much faster than both stalling and bypassing in tests such as complex mult and masked filter which contain many uses of the

muldiv unit. In cases such as vvadd where there are fewer data hazards in general, the addition of bypassing makes little to no difference. This follows from the fact that in a single core, we would not expect fewer cycles than instructions, which means the possibility of improvements dramatically falls as IPC approaches 1.

	Bin Search		Complex Mult		Masked Filter		Vvadd	
	Cycles	IPC	Cycles	IPC	Cycles	IPC	Cycles	IPC
Stall	3382	0.378179	16718	0.111497	16174	0.278162	478	0.951883
Byp	1664	0.768630	15262	0.122133	13437	0.334822	473	0.961945
Long	1664	0.768630	2512	0.742038	5658	0.795157	473	0.961945

Discussion

It is relieving to see large gains in IPC, both from bypassing and the pipelined muldiv unit. Since this processor does not affect the number of instructions and we did not change the clock period, improving IPC in these ways can improve performance with little to no reverse effect on the other factors within the Iron Law of Processor Performance. Stalling is sometimes necessary, but when bypassing is possible, this can represent huge performance improvements with little consequences in the other aspects of the architecture. However, one can imagine dangers with bypassing, especially considering the fact that the register value used is not always synchronized with the register file. In a setup where multiple processors use the same registers, for example, this could be dangerous and lead to race conditions. Of course, the disadvantages of bypassing seen through the completion of this problem set is also the added complexity in the hardware.

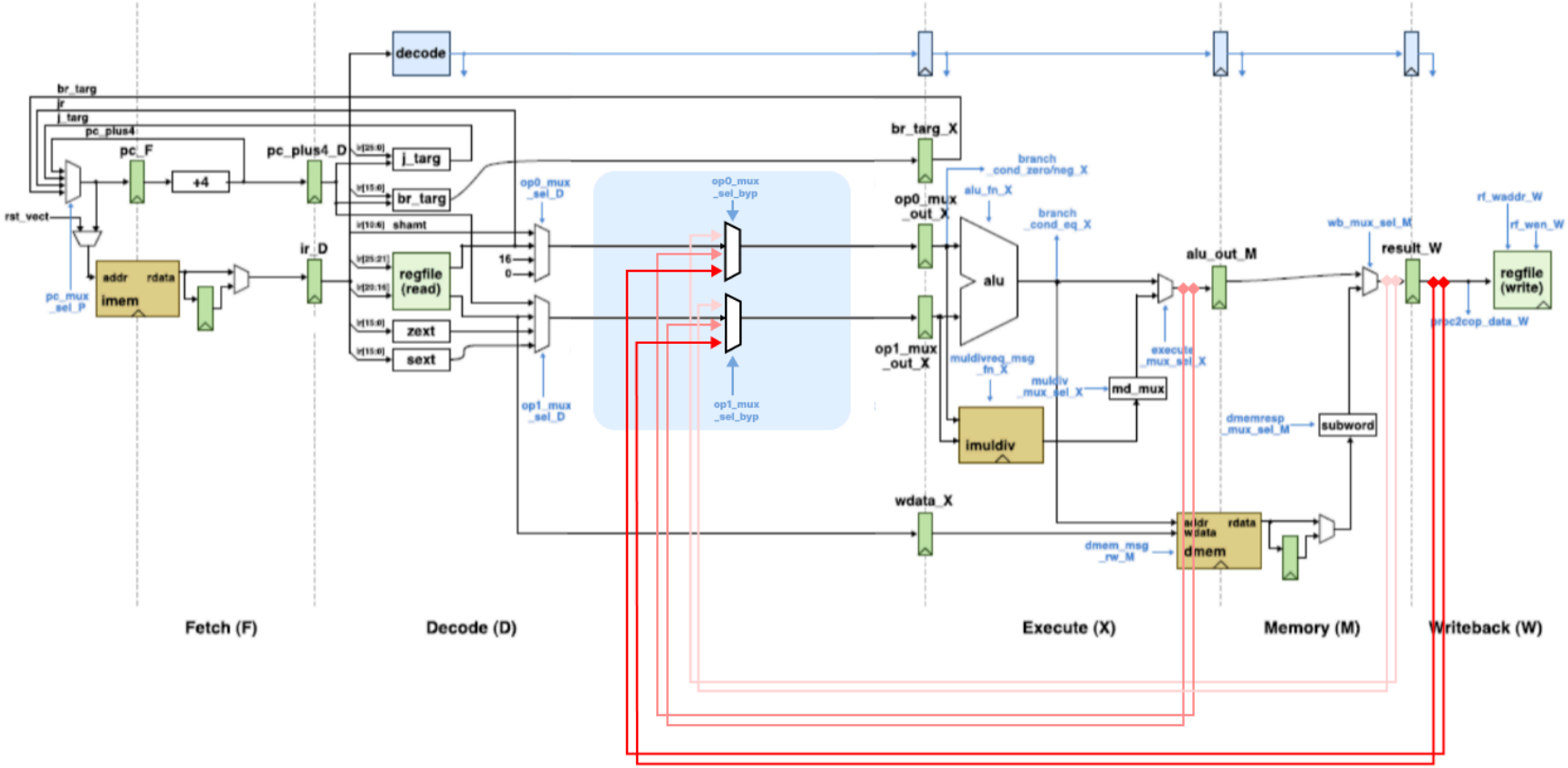

Figure 1: Datapath of 5-stage PARCv2 processor.

Figure 1. Datapath of 5-stage PARCv2 processor: This figure shows the bypass MUX for op0 and op1 after the normal MUX for these operands, but in reality, we bypass before feeding into the operand MUX. This means the jr and wdata_X wires use the output of the bypass MUX rather than the operand MUX.

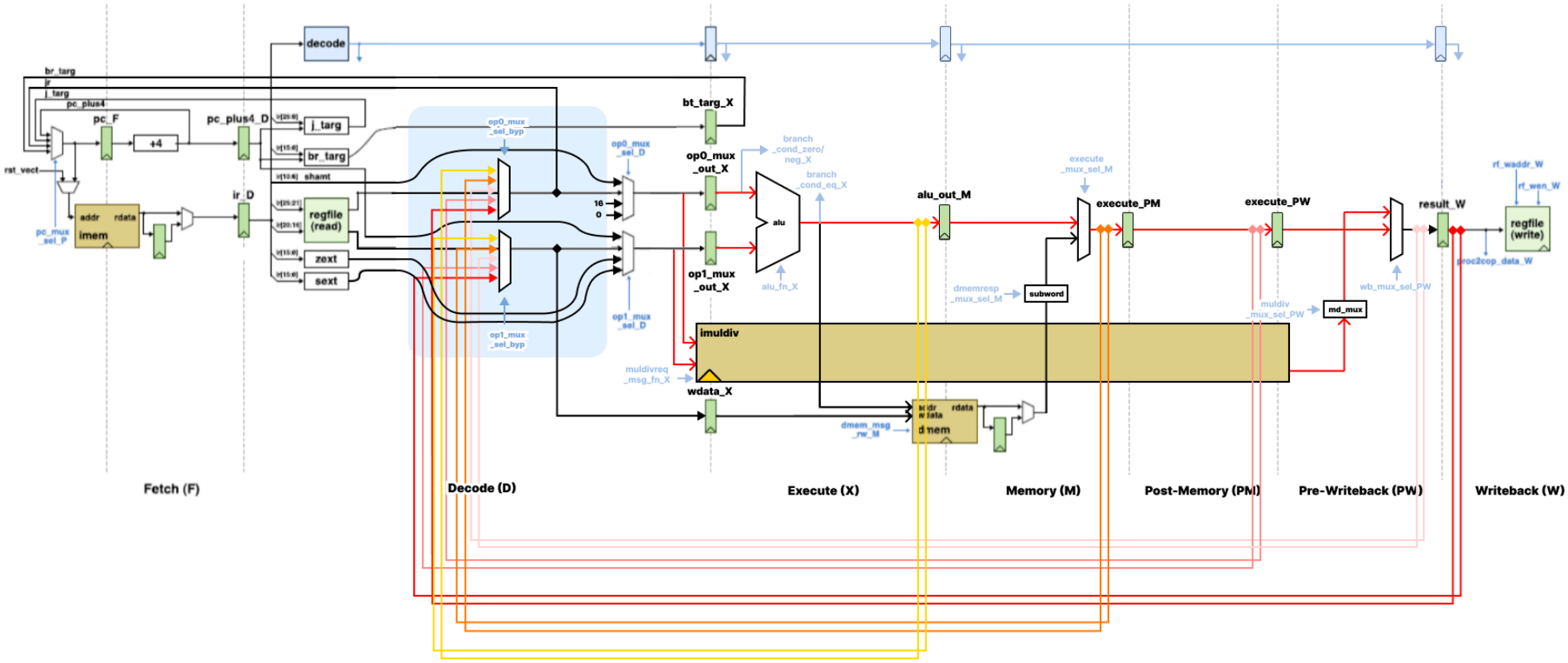

Figure 2: Datapath of 7-stage PARCv2 processor including Bypassing and MulDiv

Figure 2. Datapath of 7-stage PARCv2 processor including Bypassing and MulDiv: Please note that the order of the bypass and operand MUX's is correct here, and this is more accurate to the implementation we did in Objective 2 compared to the first figure.