1. I would select architecture A. This is because under architecture B, the Execute state would have to stall until the data was written back by the Writeback stage for the previous instruction. Under architecture A, the data is made available as soon as possible, resulting in few stall cycles and overall lower CPI. Here is an example of this (source: https://en.wikipedia.org/wiki/Operand_forwarding)

Without operand forwarding

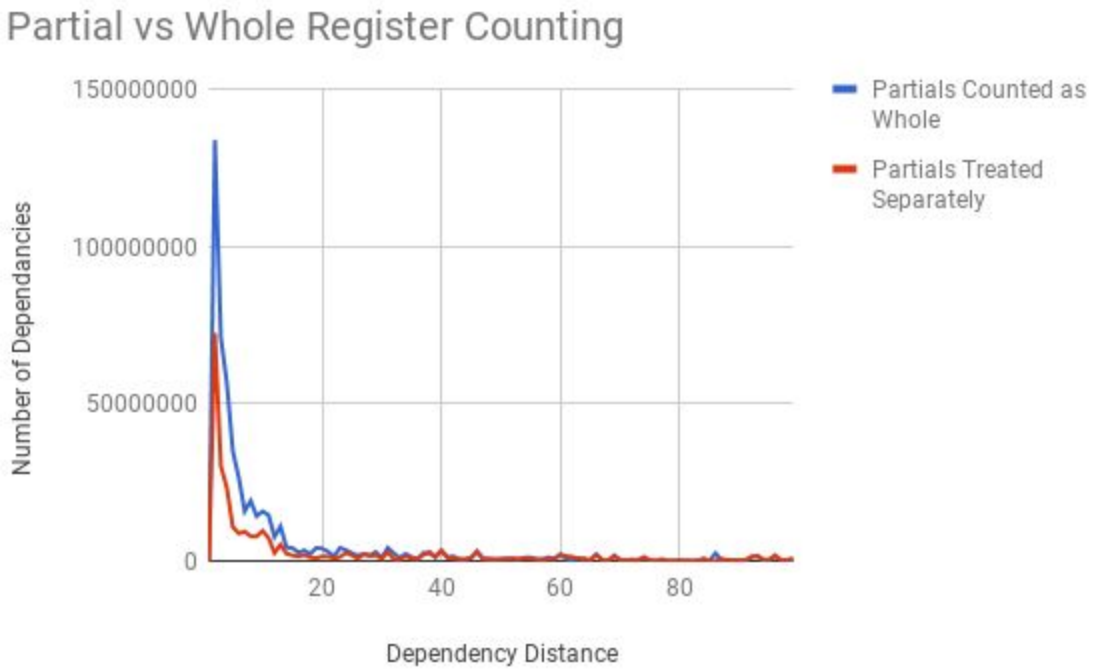
1	2	3	4	5	6	7	8
Fetch ADD	Decode ADD	Read Operands ADD	Execute ADD	Write result			
	Fetch SUB	Decode SUB	*stall*	*stall*	Read Operands SUB	Execute SUB	Write result

With operand forwarding

1	2	3	4	5	6
Fetch ADD	Decode ADD	Read Operands ADD	Execute ADD	Write result	
	Fetch SUB	Decode SUB	Read Operands SUB: use result from previous operation	Execute SUB	Write result

2. The most common registers in the tail are the scratch registers, and preserved registers. This is because they are not the primary registers used, but only done so less often. However, the general purpose registers are used often so therefore they have a shorter dependency time, meaning that few of them have high dependency lengths. Since certain registers have long range dependencies, this suggests that those particular registers do not have to have their results wired early on in the pipeline. They can have data forwarding much later on in the pipeline. This could save on the amount of wiring required and the complexity of the pipeline. However, registers that have dependencies very often will need to be be forwarded as soon as the data is made available since that is where the stall cycles will spend most of their time. (The following charts are from the black sholes test).

3. This tradeoff is is that you are more likely to catch partial register stalls, where you modify the partial register than read the whole register. However, the downside is that you will count dependencies where you read and write from the same partial register which has a lower stall cycle. So long as there are more partial register stalls than there are reading and writing from the same partial registers, the tradeoff is acceptable to make. Here is a simulation where the partial register name is read rather than the full register name (this was done by removing the function that read the full name of a register:



Notice how the number of dependencies when the partial registers are counted as whole is significantly more than if the partial registers are treated separately. This means that there would be a lot of partial register stalls, suggesting that it was a correct assumption to treat the partial registers as whole registers.

4. Architecture B has more general purpose registers because there are fewer short range dependencies. Lots of short range dependencies is usually a sign that a certain register is being overwritten multiple times for intermediate steps. Since architecture B has fewer short range dependencies, it likely has more general purpose registers to store the results of intermediate calculations.