**Report Computer Architecture 2021**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| *Implementation* | *Total Cell Area (mm2)* | *Area without data path (mm2)\** | *Critical Path (ns)* | *Maximum Operating Frequency (MHz)* | *Number of Cycles for program MULT* | *Minimal time to execute the program MULT* |
| *Single Cycle* | *242,896.1983* | *13,555.787* | *30.26* | *33.047* | *1179* | *35,676.54* |
| *Single Cycle with Multiplication Support* | *246,550.7889* | *17,203.7699* | *39.78* | *25.138* | *37* | *1,471.86* |
| *Pipelined* | *248,753.7092* | *19,414.5686* | *23.58* | *42.409* | *37* | *872.46* |
| *Pipelined with hazard and stall logic* | *248,789.7976* | *19,444.0493* | *23.54* | *42.481* | *25* | *588.5* |

*\* Since the instruction memory and data memory are practically the same for every implementation, these areas have been subtracted from the total cell area to create a better overview.*

*Questions:*

* ***For the single cycle processor, which kind of instruction would stimulate the critical path found? How would you improve it without adding any pipe stage?***

The load/store operation passes the register, ALU and memory. Mostly the memory will add a large time delay.

Improving: how? -> caching doesn’t improve critical path

* ***For the single cycle processor, which resources constitute most part of the gates? What is your explanation for this distribution? Is it possible to reduce gate cells number?***

The register, because it contains plenty of flipflops, which are built out of gates. *(Or memory because of the decoding block)*

*Gate cells = gates?*

*Reduce number of gates: less reg, different type of reg…?*

* ***What are the advantages of using a single cycle processor compared with more advanced implementations? Can you imagine/propose an application scenario of such cores?***

Smaller surface area and simple design and therefore cheaper. This can be useful for simple applications. (And more efficient? Because of no NOPs and less capacitance)

* ***Is the critical path affected when hardware support for multiplication is added to the single cycle processor? What is your explanation for this? Do you know any multiplier implementation that can improve timing?***

Yes, the critical time increases. This because the ALU becomes slower (since extra hardware is needed for multiplication support) and therefore also operations that use the ALU such as calculating the addresses for load/store.

Implementing a separate multiplier could avoid slowing down addition operation used in the critical path. (but higher fan out?)

Should we look for other algorithms?

* ***Is adding hardware support for multiplication a good choice for every microprocessor? Motivate exhaustively your answer.***

No. You should only include a multiplier if multiplications are a significant part of the total of ALU operations. Otherwise, you needlessly slow down the other operations only to speed up a few multiplications. Also, costs increase when adding a multiplier. This is only worth it if the time reduction is useful. In simple appliance such as a coffee machine the critical path is of low importance. Also the multiplication ALU might use more power per cycle.

* ***How much larger is the pipelined implementation compared to the single cycle processor? What is the main cause for its increase? How is the critical path affected when we pass from a single cycle processor to a pipelined implementation ?***

The area of the CPU, when not considering the data and instruction memory, increases from 17,203.7699 mm2 to 19,414.5686 mm2. This is mainly because of the addition of extra registers. The critical path is greatly reduced from 39.78 ns to 23.58 ns because it now only spans one stage of the processor (between two registers) instead of the whole operation.

* ***Taking into account the critical path found for the pipelined processor, how would be possible to increase the performance of the system? Would your solution significantly speed up the core? Also, what will be the new critical path?***

Adding extra caching stages can improve the performance of the load and store operations by improving the average access time. This will not improve the critical path however, since a cache miss will still result in a high (even slightly higher) access time.

* ***What other microarchitecture techniques (besides pipelining) could be implemented in our microprocessor in order to improve performance? Please explain under what conditions/type of workload you will have the maximum/minimum performance.***
  + Multi-threading: The compiler becomes more complicated and extra logic is added. As a result, only if enough instructions can be shifted to increase performance, the overall performance will increase. Otherwise, the overhead will only slow down the processor without producing any time gain.
  + Parallelizing: Running multiple operations in parallel by replicating hardware such as a whole core in a mulit-core implementation or individual components.

The performance depends on the program and especially on the dependencies in the program.

* + Pipelining: if you need to stall the pipeline a lot because you are waiting for data to arrive, the performance will be less.
  + Multi-threading: only a way to resolve dependencies and to already start running other code if there are too many dependencies.
  + Parallelizing: if you have multiple blocks on code that can run in parallel without depending on each other.
* ***Is the addition of hardware improvements, like pipelining, correlated with higher power consumption? How can we assess if a specific modification to our processor improve or diminish the energy efficiency of the system?***

It depends on the type of instructions processed and their frequency if the power consumption increases or diminishes. Standard benchmarks can provide more insight.

Although the power consumption might increase when adding more components, the pipeline implementation is generally faster and will therefore use less energy if the power is the same.