Selection of the Optimum Stage Number in Pipelined Floating-Point Units

Eduardo Balliriain, Martín Ignacio Falcón Faya, Pablo Slavkin, Norberto M. Lerendegui*
Instituto Tecnológico de Buenos Aires, Departamento de Electrónica,
Av. E. Madero 399, (C1106ACD) Buenos Aires, Argentina
*Corresponding Author (nlerende@itba.edu.ar)
Keywords: Floating Point Unit, Pipelining, Computer Architecture

Abstract

In this work the pipeline theory applied to computing systems is reviewed. The effects of the stage delay, overhead stage delay, equalization factor and number of stages on the pipeline system performance are analyzed. A pipeline design method to identify the optimum number of stages is proposed. This method makes use of a trade-off expression that considers speed factor and hardware cost. The procedure is applied to turn a sequential Floating Point Unit (FPU) into a Pipelined Floating Point Unit (PFPU) capable to achieve a performance 600% larger. The effect of the physical limits on the PFPU maximum performance is analyzed.

1. Introduction

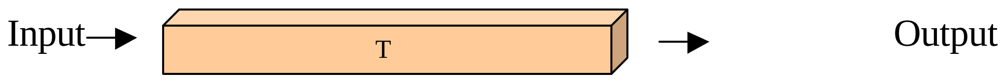
The idea of splitting up any sequential process in stages in order to increase the performance has been around for many years. Henry Ford applied this concept successfully in his Ford-T model assembly line, increasing the number of units produced per day and making the Ford-T the most famous car of its time.

This splitting method, named ***pipelining***, was further applied to processor architecture; initially to enhance the processing speed in the classical fetch-decode-execute cycle [1], and then whenever a sequential process is detected. Pipelining became a must-do approach when searching for better performance.

In this work the pipelining theory and implementation are presented. A method to identify the optimum number of stages is proposed. This method is further applied to turn a sequential Floating Point Unit (FPU) into a Pipelined Floating Point Unit (PFPU). The processing speed and the hardware size for both implementations are compared.

2. Pipelining Theory

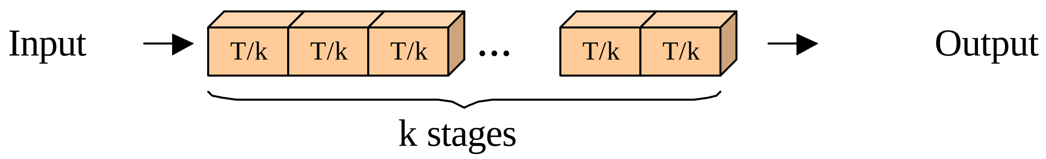
Given an inherently sequential process that consumes an amount of time T, once the input has started being processed the next input has to wait until the ongoing process finishes completely.



Two important parameters are:

- <u>Time to Process a single Input (Latency):</u> Once an input is pushed into the system, it takes a time T to get the output available. This happens for every input.
- <u>Time to Process N Inputs:</u> If N inputs are entered to the system, it takes N*T to process the entire input set.

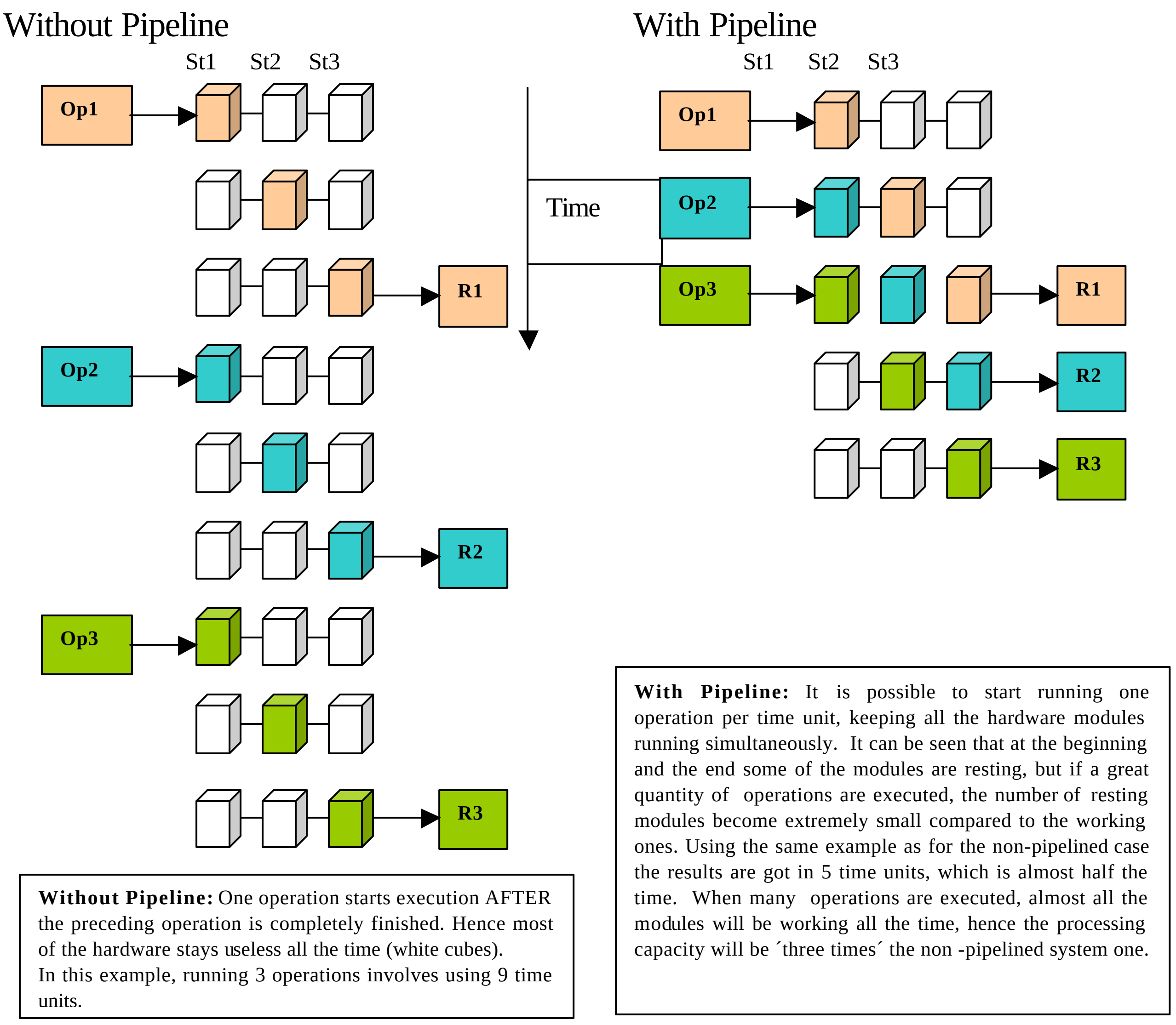
If the whole process is symmetrically splitted up in k stages (meaning that each stage consumes a T/k time), then each time one stage delivers its output to the next stage, the former is free to take a new input in. The number ***k*** is known as the ***order*** of the pipeline.



When the first input of the set is fed to the system, it takes a time T for the output to be available (because the input has to pass all the processing stages). But things are now different for the following inputs. While one input is being processed at the stage St_j the following input is being processed at the stage St_{j-1} (previous stage). Therefore, once the first system output is available the next one will be ready a time T/k later. Theoretically, there is no lower limit for the stage delay as long as the order of the pipeline is increased to the required level. As a consequence, if N inputs are sequentially fed to the pipelined system, the processing of the entire input set takes:

$$\underbrace{T}_{\text{First Input (Latency)}} + \underbrace{\frac{T}{k}\cdot(N-1)}_{\text{Following N-1 Inputs}}$$

The **Figure 1** illustrates a 3-Stage Operation Unit without and with Pipelining



Without Pipeline: One operation starts execution AFTER the preceding operation is completely finished. Hence most of the hardware stays useless all the time (white cubes).
In this example, running 3 operations involves using 9 time units.

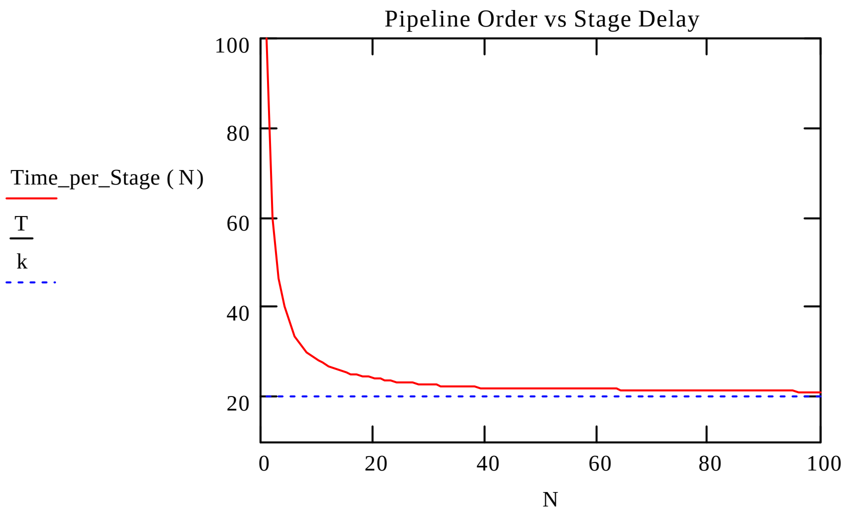
With Pipeline: It is possible to start running one operation per time unit, keeping all the hardware modules running simultaneously. It can be seen that at the beginning and the end some of the modules are resting, but if a great quantity of operations are executed, the number of resting modules become extremely small compared to the working ones. Using the same example as for the non-pipelined case the results are got in 5 time units, which is almost half the time. When many operations are executed, almost all the modules will be working all the time, hence the processing capacity will be ´three times´ the non -pipelined system one.

Figure 1: Non-Pipelined and Pipelined 3-Stage Operation Unit

If N is very high the time it takes to the system to produce the entire set of N outputs can be approximated by:

$$T + \frac{T}{k} * (N - 1) \cong \frac{T}{k} * N$$

indicating that in average each input will be processed in T/k units of time. The **Figure 2** shows the Input Average Processing Time (Time_per_Stage) as a function of N (number of inputs to process) when T=100 and k=5. When N=1 the Time_per_stage is equal to T.



When N is small, the latency of the system can strongly affect the average processing time.
When N is high the average time to process an input tends to T/k.

* Note that the pipeline concept requires that all stages have the same processing time (Tst); one stage output can not progress if the next stage is busy.

Figure 2: Input Average Processing Time as a function of the Number of Inputs to process

3. Pipelining Implementation

The Pipelining technique consists of identifying any sequential stage in a process and organizing the hardware in stages to maximize its performance. In practice, two factors negatively affect the pipeline efficiency:

1) Although all stages in a pipeline must have the same length, this will be seldom achieved naturally in real world applications. This problem is solved by forcing all stage processes to last as the slower one (equalization). As presented in the **Figure 2**, this time, named ***stage delay***, represents the average time taken by the system to produce every output when N is high enough. The lower the stage delay, the higher the system speed. The **Figure 3** shows the stage equalization process.

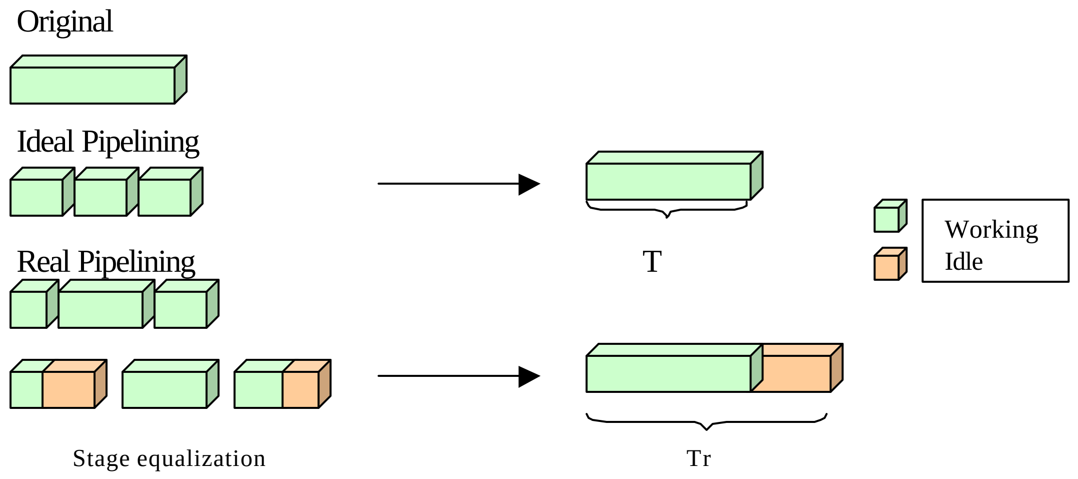


Figure 3: Stage Equalization Process

After the system has been equalized, the new latency Tr will be larger than the initial one (T). The ***Equalization Ratio (EQR)*** can be defined as the ratio between the Equalized Stage Delay and the Ideal Stage Delay:

$$\mathrm{EQR} = \frac{\frac{Tr}{k}}{\frac{T}{k}} = \frac{Tr}{T}$$

2) Creating the stages not only has an associated increase in hardware to hold a stage output while the next stage is still processing, but also adds an extra fixed delay to each stage, named ***Overhead Delay (OD)***, related to that extra hardware. The **Figure 4** illustrates this ***Overhead Delay***.

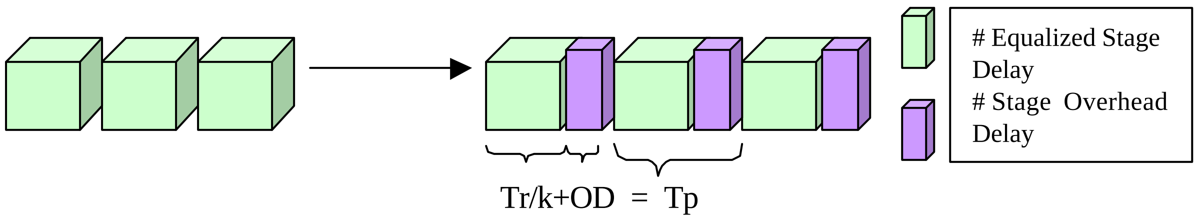


Figure 4: Stage Overhead Delay

The ***OverHead Ratio (OHR= OD/Tp)*** indicates how much of the stage time is wasted in the associated hardware (note that 0< ***OHR*** <1). When OHR is close to 1 the system spends most of its time holding data without processing it. On the contrary, the nice empirical value OHR=0.001 indicates that 99.9% of the Stage Delay is used for processing.

The former two factors determine that the Stage Delay in a real system be:

$$StageDelay = \frac{Tr}{k} + OD$$

It is clear that the term Tr/k diminishes when the order of the pipeline (k) increases, while the term OD remains constant. When Tr/k has a similar magnitude than OD the increasing of the pipeline order will not decrease the stage delay significantly. For instance, if T=100 , OD=0.1T and EQR=1.1, the Stage Delay will be:

$$StageDelay = \frac{1.1 * T}{k} + 0.1 * T = \frac{110}{k} + 10$$

If k= 20, StageDelay=15.5; if k=22, StageDelay=15. In other words, a 10% increase in the stage number caused only a 3.2% decrease in the StageDelay.

4. Real World Pipeline Design

The procedure described in the former section was applied by the authors to design a Pipelined Floating Point Unit (PFPU). The PFPU is capable to perform the four basic operations (one at a time): addition (or subtraction), multiplication and division, according to IEEE 754 standard for single precision [2][3][4]. The original FPU was designed without pipeline [5]. The **Figure 5** shows a simplified diagram of the PFPU architecture. Information concerning hardware delays and number of gates was collected in order to make a clear comparison of the performances of the Floating Point Unit with and without pipeline.

In order to make the analysis independent of the hardware technology used and extend its conclusions to other designs, all delays are represented in units of 'Transistor Delay Time' (Td), which is defined as the time it takes to a transistor to produce a stable output once a stable signal is present at its input. This time depends on the chosen IC technology but it is fully documented by hardware manufacturers. All delays shown in this work were obtained by simulating the designed hardware with the commercial OrCAD V.9 software product. The original PFPU operation blocks has the following processing times:

Adder: 152 Td Multiplier: 48 Td Divider: 422 Td

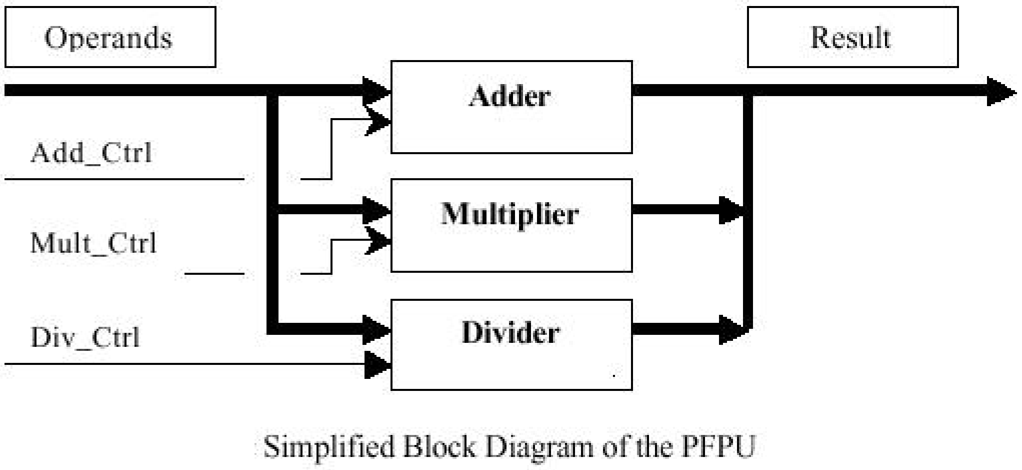


Figure 5: PFPU Simplified Architecture

This design takes into account two main factors to decide the number of pipeline stages to implement:

1) <u>Processing Speed:</u> according to pipeline theory, increasing the number of stages while keeping low OHR figures makes the FPU run faster
2) <u>Hardware Cost (measured in quantity of digital gates used):</u> A larger number of stages increases the hardware cost of the system. There is an average quantity of transistors for all digital gates involved in this design. This value is used to assess the hardware cost.

These two factors drives the designer to take a critical implementation decision. Therefore, it is convenient to find out some analytical tool to assess the trade-off.

The procedure of choosing the optimum pipeline stage number requires that the hardware circuit be carefully studied in order to propose possible "cut points" in the circuit to form the different pipeline stages. Every time a pipeline stage is added, the stage delay is decreased (see **Figure 6**), but a quantity of latches proportional to the quantity of data lines cut must be also added.

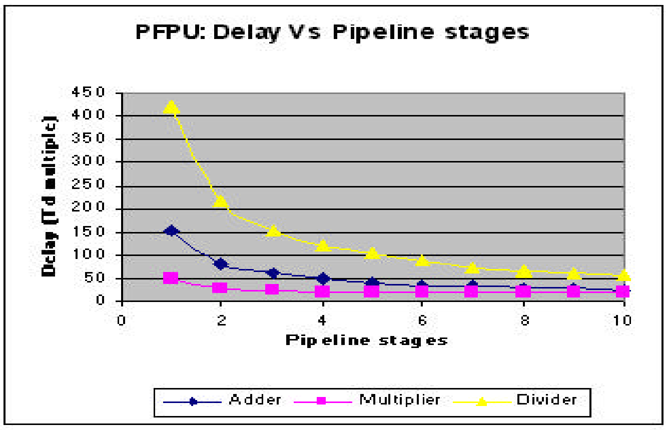
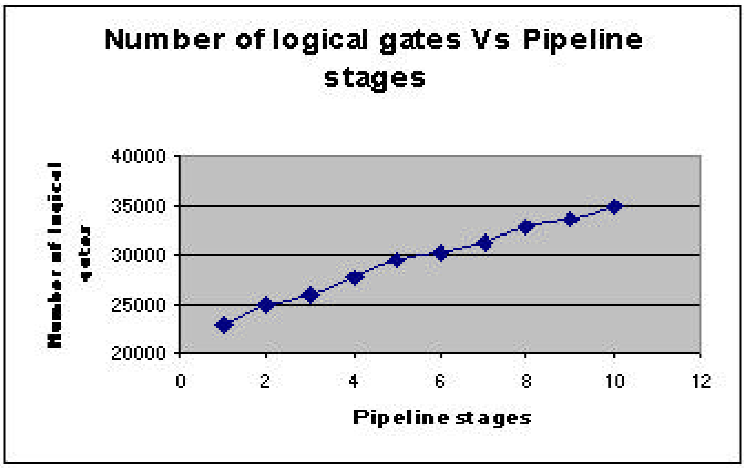


Figure 6: PFPU Stage Delay as a function of the number of stages

This quantity of latches has a very wide range of variation, since the system may be cut in a place where 5 data lines have to be latched or in a place where 300 data lines must be latched.

The fact that there are certain places where the quantity of latches added is too high forces the designer to cut the system before or after that place, increasing the difference of length between stages and making the total stage delay bigger after the equalization. The criteria followed by the authors is that the quantity of latches added must grow linearly with the number of stages added. The circuit was studied to propose cutting points to implement a pipeline of order 1 to 10. The **Figure 7** represents the number of logic gates as a function of the pipeline stages.



A careful selection of the points in the circuit to cut, leads to an almost linear growth of the quantity of latches employed as the pipeline order increases.

Figure 7: Number of Logic Gates as a function of the Pipeline Order

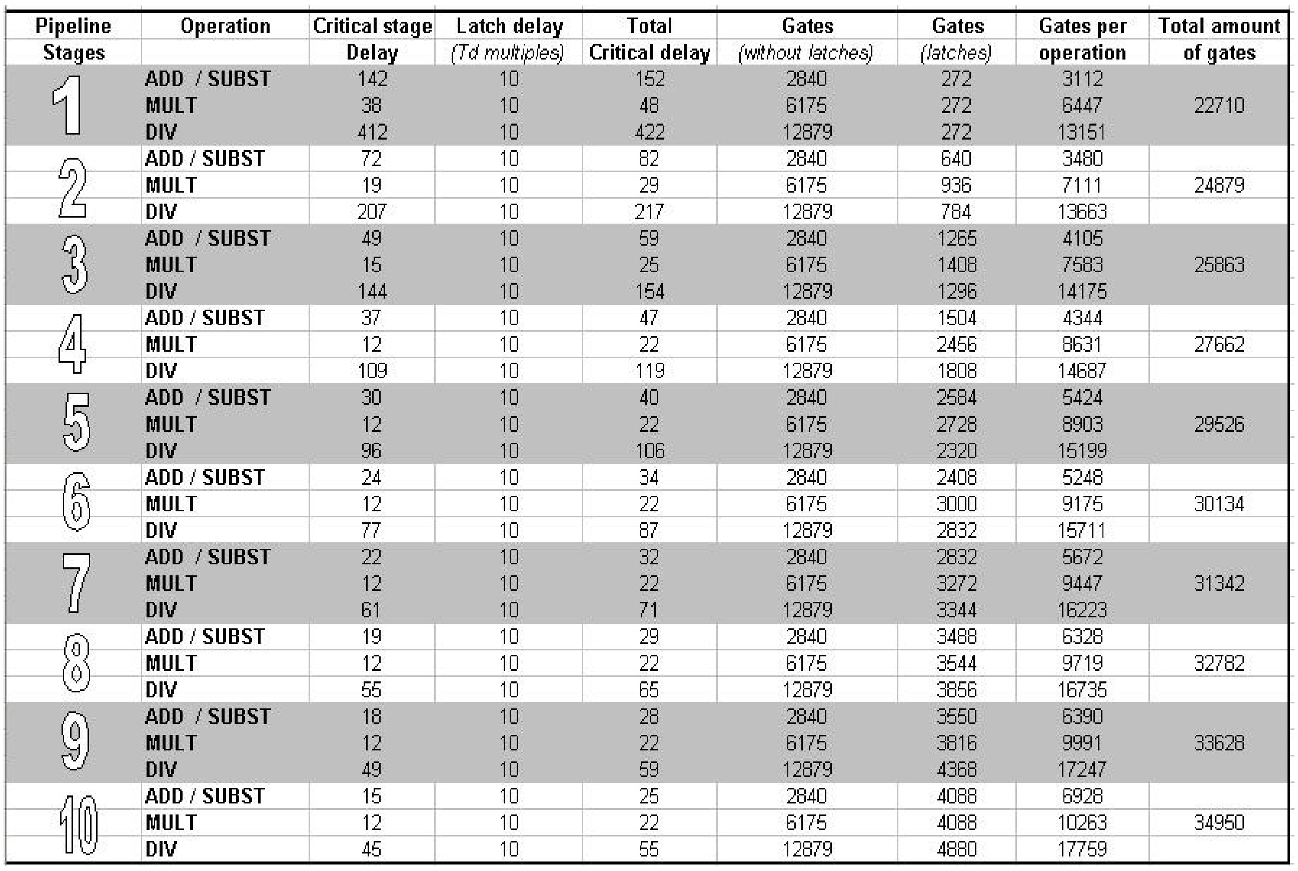
For all operation modules, the slower data path was found (critical path) to determine the worst case delay for each stage (critical delay). Since the divider block causes the largest critical delay for all stage numbers considered and the system was designed to work synchronously, this delay is the chosen one for this design. The latches used in this design add a constant delay equal to 10 Td to each stage of the pipeline. The data collected is summarized in the **Table 1**.

The complete pipeline design procedure applied to the PFPU is the following:

A) <u>Gathering of information concerning the possible places where ´cuts´ to introduce latches could be made (cases k= 1 to 10 analyzed).</u> As these ´cuts´ would leave asymmetric stages (e.g. it is not a good idea to insert latches inside a full adder, since too much hardware would be involved), the time taken by the stages differs from one another. Therefore the ´critical path´ concept, defined as the longest time taken for any stage to fulfill its task, becomes very useful. As expressed above, this time is measured using transistor delay times (Td) as normalized units. Independently of the number of stages taken, the time used by all stages is completely defined by the divider circuit due to the fact that this circuit is much slower than the multiplier and adder. As a consequence, both adder and multiplier will have idle time.

B) <u>Calculation of the critical path for all 10 pipeline orders taking into account that each time a pipeline stage is added, a latch delay (overhead) must be added to the stage delay.</u> The memory cells used in this design have a time response delay of 10*Td. Note that the stage time falls nonlinearly with the increasing number of stages (see **Figure 6**). This is due to the fact that the shorter the stage the stronger the effect of the constant delay (overhead) of the latches (10*Td per latch). As shown in the **Table 1**, for a 10-stage PFPU the 22 % of the delay is due to the latches.

C) Counting of the number of gates required for all ten pipeline orders to get an idea of the ´hardware cost´ involved in each case. The number of logical gates used grows **linearly** with the number of pipeline stages to implement (because the places of the circuit cuts were chosen carefully).



Pipeline Stages	Operation	Critical stage Delay	Latch delay *(Td multiples)*	Total Critical delay	Gates *(without latches)*	Gates *(latches)*	Gates per operation	Total amount of gates
1	ADD / SUBST	142	10	152	2840	272	3112	
	MULT	38	10	48	6175	272	6447	22710
	DIV	412	10	422	12879	272	13151	
2	ADD / SUBST	72	10	82	2840	640	3480	
	MULT	19	10	29	6175	936	7111	24879
	DIV	207	10	217	12879	784	13663	
3	ADD / SUBST	49	10	59	2840	1265	4105	
	MULT	15	10	25	6175	1408	7583	25863
	DIV	144	10	154	12879	1296	14175	
4	ADD / SUBST	37	10	47	2840	1504	4344	
	MULT	12	10	22	6175	2456	8631	27662
	DIV	109	10	119	12879	1808	14687	
5	ADD / SUBST	30	10	40	2840	2584	5424	
	MULT	12	10	22	6175	2728	8903	29526
	DIV	96	10	106	12879	2320	15199	
6	ADD / SUBST	24	10	34	2840	2408	5248	
	MULT	12	10	22	6175	3000	9175	30134
	DIV	77	10	87	12879	2832	15711	
7	ADD / SUBST	22	10	32	2840	2832	5672	
	MULT	12	10	22	6175	3272	9447	31342
	DIV	61	10	71	12879	3344	16223	
8	ADD / SUBST	19	10	29	2840	3488	6328	
	MULT	12	10	22	6175	3544	9719	32782
	DIV	55	10	65	12879	3856	16735	
9	ADD / SUBST	18	10	28	2840	3550	6390	
	MULT	12	10	22	6175	3816	9991	33628
	DIV	49	10	59	12879	4368	17247	
10	ADD / SUBST	15	10	25	2840	4088	6928	
	MULT	12	10	22	6175	4088	10263	34950
	DIV	45	10	55	12879	4880	17759	

Table 1: Number of Logic Gates and Total Critical Delay as a function of the Pipeline Order in the Pipelined Floating Point Unit

D) Normalizing of all of the collected data (forcing it to be between 0 and 1) in order to be used in an optimization formula. This formula indicates how good the decision of using a determined order of pipeline is. This formula takes into account the gain in speed related to a higher order of the pipeline, the increase in hardware cost and the relative weight the designer allocates to these factors. The optimization formula was defined as:

$$\mathbf{Trade\_Off_k} := \frac{1}{(1 - \mathbf{P}) \cdot \mathbf{NormGates_k} + \mathbf{P} \cdot \mathbf{NormDelay_k}}$$

Where: $$\mathbf{NormDelay_k} := \frac{\mathbf{Total\_Delay_k}}{\mathbf{max(Total\_Delay)}} \qquad \mathbf{NormGates_k} := \frac{\mathbf{Total\_Gates_k}}{\mathbf{max(Total\_Gates)}}$$

P is the Speed Weight (real number between 0 and 1).
k (pipeline order) is an index equal to the number of stages.

In the PFPU design the speed was considered more important than the hardware cost. Speed Weight was 60% and Hardware Weight 40%. With these parameters the former expression becomes:

$$\mathbf{Trade\_Off}_k := \frac{1}{0.4 \cdot \mathbf{NormGates}_k + 0.6 \cdot \mathbf{NormDelay}_k}$$

When the Trade-Off expression is depicted as a function of k (see **Figure 8**) it is possible to identify the maximum of the function. Its corresponding k-coordinate (7, in this case) represents the best pipeline order for the PFPU regarding the assumptions stated.

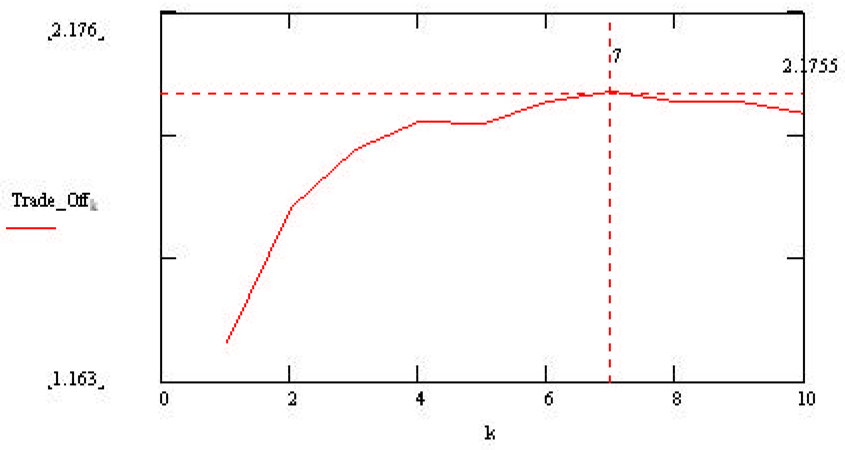


Figure 8: PFPU Trade-off Evaluation Figure as a function of the Pipeline Order

5. Comparisons and conclusions:

The **Table 1** shows that in the case of a first order pipeline (placing 1 series of latches at the output of the operation blocks to hold the results calculated), the critical delay is 422*Td. The same system with a seventh order pipeline structure has a critical stage delay of 71*Td, which represents a 600% increase in the operation execution throughput. If the latches added to the system did not increase the stage delay the increase in processing throughput would be 700%, as predicted by the pipeline theory.

The Trade-Off formula takes into account that increasing the pipeline order does not necessarily lead to better performance. In this design the best choice was a 7-stage pipeline architecture.

While the increase of the pipeline stage delay has an intrinsic non-linear behavior (because of the constant value of the overhead), the increase in hardware caused by the growing of the pipeline order can be forced to be linear by making an intelligent selection of the ´cuts' in the circuit. This fact guarantees the scalability of the hardware associated.

There is a well defined limit for the increase in speed performance in a real world pipeline design, as shown in the Delay/Pipeline Stage curve. This limit appears because as the stage delay becomes smaller (the order of the pipeline grows), it becomes comparable to the overhead delay, that remains constant regardless the pipeline order.

6. References:

[1] A. S. Tanenbaun, ***“Structured Computer Organization”,*** Fourth Edition, Prentice Hall,1999.

[2] ***“IEEE Standard for Binary Floating-Point Arithmeti***c”, ANSI/IEEE Standard 754, 1985.

[3] D. Goldberg, ***“What Every Computer Scientist Should Know About Floating-Point Arithmetic***”, Xerox Palo Alto Research Centre, 3333 Coyote Hill Road, Palo Alto.

[4] W. Kahan, ***“IEEE Standard 754 for Binary Floating Point Arithmetic, Lecture notes on status of IEEE 754***”, May 31, 1996.

[5] E. Balliriain, M. Falcón, P. Slavkin, "***Diseño y simulación de una unidad de punto flotante conforme a la norma IEE-754***", Final Project, Electronics V, ITBA 2002.