Design and Simulation of Four Stage Pipelining Architecture Using the Verilog

Darshil Mavadiya (22BEC508)
Dept. of Electronics and Communication Engineering
Institute of Technology, Nirma University)
Ahmedabad, India
22bec508@nirmauni.ac.in

Kirti Bhatt(22BEC503
Dept. of Electronics and Communication Engineering)
Institute of Technology, Nirma University
Ahmedabad, India
22bec503@nirmauni.ac.in

***Abstract*—The idea of parallelism is used by computers and other devices to accelerate system activities. The pipelining notion is one of the parallelism techniques. Pipelining is being used by many devices to boost throughput and speed. There are several phases that make up the overall pipeline stage, including fetch, decode, execute, and store. In this work, the Quarts and Modelsim simulators are used to independently construct and simulate a four-stage pipeline. It illustrates how the procedures are carried out at each pipeline stage.**

***Index Terms*—instruction, pipeline, speed of operation, processor cycle**

I. Introduction

As the name suggests, an instruction tells the computer what to do. Put simply, each command we as users enter into a program tells the computer to carry out a set of actions. These high-level instructions are translated by a computer into machine-understandable code, or machine language, which consists of one and zeros. An instruction breaks up the work to be done into manageable segments, or pieces, that each take a fraction of the time to finish the instruction as a whole. A pipe stage (or pipe section) is any one of these steps. To create a pipe, pipe stages are joined. A computer's fundamental cycle of operation is called an instruction cycle. It involves a computer retrieving a program instruction from memory, figuring out what steps need to be taken to complete the instruction, and then taking those steps. From the time the computer boots up until it shuts down, the central processing unit (CPU) keeps repeating this cycle. The instruction cycle is carried out sequentially in simpler CPUs; one instruction is finished processing before the next is started. The majority of contemporary CPUs, on the other hand, execute the instruction cycle concurrently in parallel, forming an instruction pipeline. This is made possible by the cycle's division into discrete parts, which allow the processing of the subsequent instruction to begin before the preceding one is completed. Since the 1960s, pipelining computer architecture has drawn a lot of interest as the demand for faster and more affordable systems became urgent.The benefit of pipelining is that it can assist in matching the pace of different subsystems without increasing the overall system cost.The availability of LSI circuits at lower costs and quicker speeds has made pipeling, whether in simple or sophisticated forms, more viable in the future. Overlapping operations as they are being executed is known as pipelining. These days, this is a crucial component of quick CPUs. There are various kinds of pipelines, including multi-issue, operation, and instruction pipelines. We will mimic the fetch, decode, execute, and store phases of pipelining in this work. To demonstrate how each step functions, a separate simulation can be run. The Quarts tool and Modelsim simulator can be used to carry out the simulation process.

II. Concept of pipelining

Pipelining, a feature that allows many instructions to be executed simultaneously (that is, during a single clock cycle), is built into RISC processors by default. In a pipelined processor, each datapath stage is isolated from the others. Each stage in the datapath takes an input from the previous stage, processes it in accordance with a preset protocol, and then moves on to the next step before sending the resultant output to it.

We call these operations "pipe stages" or "pipe segments." The steps are connected to form a pipe, via which instructions enter at one end, progress through, and exit at the other. The throughput of an instruction pipeline is determined by how frequently an instruction exits it. The time required to move an instruction one step down the pipeline is known as a processor cycle. Since each stage occurs simultaneously, the slowest pipe stage's duration determines the length of a processor cycle; so, the longest step would define the interval between advancing pipe stages. The processor cycle of a computer usually consists of one clock cycle, but it can occasionally consist of two or more. The number of pipe stages multiplied by the pipelining-induced speed gain in these conditions. Consequently, the time per instruction of the pipelined processor may approach its lowest value, even though it won't attain it.

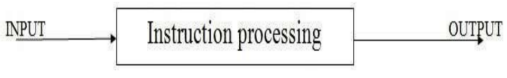

Fig. 1. Processing of instruction

The average execution time per instruction is lowered as a result of pipelining. It is possible to interpret the reduction

as a decrease in the clock cycle time, reduction in either the clock cycle duration, the clock cycles per instruction (CPI), or both. If the processor at the beginning of the pipeline needs several clock cycles to complete an instruction, this is usually interpreted as reducing the CPI. We will primarily take this position. In processors designed to execute instructions in a single, extended clock cycle, pipelining reduces the clock cycle time. An instruction's transition from one step to the next takes time, measured in machine cycles, which are usually represented by one clock cycle. Throughout several machine cycles, a single instruction must be carried out via the pipeline.When all N stages have run for a set period of time (N-1 cycles), the pipeline reaches its maximum capacity. Currently, N instructions are being executed simultaneously by the pipeline at maximum parallelism. It appears that larger stages are always associated with better performances. However, synchronization and information transfer between stages get more complex with an increase in the number of steps. The more steps, the more sophisticated the CPU becomes. Pipelining is the process of breaking up instructions into smaller tasks for instruction execution, such as fetch, decode, execute, and store result.Each of these smaller jobs is completed by a pipeline stage, which generates interim results that need to be saved in order for an instruction to move on to the next step. If execution is broken down into more manageable subtasks, then the different subtasks of several unique instructions may overlap simultaneously.

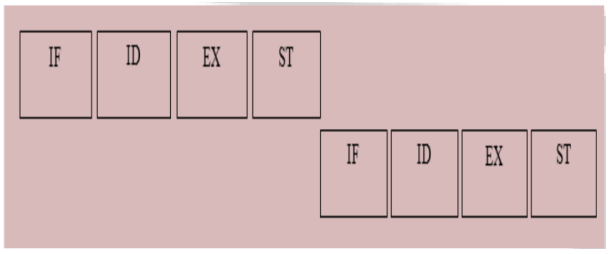

Fig. 2. Unpipelined processor

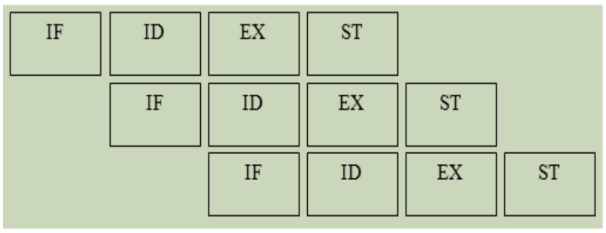

Fig. 3. Pipelined processor

A. Performance of pipelining

The number of instructions performed in a given amount of time, or CPU instruction throughput, is increased by pipelining; nevertheless, the execution time of a single instruction is not shortened. Actually, because of pipeline control overhead, it typically results in a little increase in the execution time of each instruction. An increase in instruction throughput makes a program operate faster overall, even if no single instruction executes more quickly. A pipeline has practical depth constraints since the execution time of an instruction never decreases. Pipeline latency, pipelining overhead, and imbalances between the pipe stages are the main causes of limits. Performance is hindered by an imbalance between the pipeline stages since the clock cannot run faster than the time needed for the slowest pipeline stage. For CPUs that are not pipelined, the clock per instruction may be elevated. On the other hand, pipelining speeds up execution while decreasing CPI.

B. Use of Pipeline

Assuming we follow a hundred instructions. Single-cycle machine: 41 ns/cycle × 1 CPI x 100 initializations = 4100 ns. Machine with multiple cycles: 10 ns/cycle × 4.1CPI (due to the initial mix) x 100 initial = 4100 ns. A pipelined machine that is optimal would be 10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns.

C. characteristics of a pipeline

Latency is the length of time it takes to execute a single action. The pipelined processor's delay determines how quickly dependent instructions can be completed. Throughput is the pace at which operations are completed. It is commonly expressed as operations/seconds or operations/cycles. Throughput ¿ 1/la-tency is the outcome of overlapped instruction execution. Pipelines result in fewer cycles per instruction (CPI). A processor cycle is the amount of time required to advance an instruction one step at a time down the pipeline. A CPU cycle is equivalent to one CPU clock cycle. Consequently, the slowest pipeline stage determines the length of the processor cycle. The purpose of pipelining is to lower the average execution time per instruction (CPI). The CPI of a pipeline should ideally match the CPI of each pipeline step. Pipestage count is equivalent to the potential speed boost. Uneven pipe stage lengths result in decreased speedup.

III. Phases of Pipeline

An instruction is carried out in four steps. A few of them include instruction fetch (IF), execution (EX), store (ST), and instruction decode operand fetch (ID). We used a register transfer notation to describe each step of the instruction execution process.

A. Instruction fetch (IF)

This stage involves using the PC (program counter) as the address of the memory location to acquire the next instruction to be executed from memory (read the instruction from memory). Instruction register (IR) is assigned to the matching position that is accessed in the (instruction) memory. Data Recovery Memory [PC]; Program counter increases signal the start of the following instruction. This is accomplished by adding one to the current program counter. Within the NPC register is the most recent value for PC. One NPC PC + one;

B. Instruction Decode Operand Fetch (ID)

Following the instruction's loading into the Instruction Register, operand fetching (decoding the instruction and retrieving the source operand(s)) begins. Opcode is used in conjunction with the function code fields to decode instructions, depending on whether they are in R, I, or J format. Using the Rs1 and Rs2 fields of the instruction, the two source operands are fetched simultaneously. The operands that were fetched are kept in internal registers A and B. Even though some instructions may have one or zero source operands, operand fetch occurs even before the instruction decoding, fetching operand values into A and B registers for all three instruction types (R, I, and J). A ¡– Rs1; B ¡- Rs2;

C. E Execution (EX)

Complete the task that the directive specifies in this phase.

The following is how arithmetic commands are carried out. The ALU receives the operands (A and B register for R-format, and A and Imm for I-format) together with the proper control signal to carry out the arithmetic operation. The opcode and function code fields of the instruction in IR are used by the control unit to produce the control signals for the ALU and the MUX that are used to select the second operand (between B and 1mm). So,R-format instruction for

$$ALU_out <-- A(+)B(forinstructioninR-format)$$

$$ALUo_ut <-- A(+)Imm(forinstructioninI-format)$$

Only the effective address of the memory location is calculated during the EX phase, not the data transfer instructions. For instance, when a LOAD instruction is used, LD R2, 8 (R1) During the decode and operand fetch phase, the source register (R1) in A was fetched, and the displacement was sign-extended in the Imm register. . Control instructions (branching (by PC relative address)) compute the branch destination address by multiplying the value of the Next Program Counter (NPC) by the offset (provided in the offset field of the instruction, loaded and sign extended in the Decode phase in the Imm register). In other words,

$$NPC + Imm <-- ALU_out$$

In this case, the operand from the A register is multiplexed with the NPC operand as the first input.

D. Store (ST)

The result value is written in the destination register (save the result in the destination location) for the ALU and load instructions during this phase. When a conditional branch instruction is given, the PC is loaded with either the branch target address (determined in the EX phase) or the NPC, depending on whether the condition is true. For teaching ALU,

$$Rd - LDR <-- ALU_outForLDinstruction$$

The computer is run by a clock whose duration allows every instruction's retrieve, decode, execute, and store steps to be finished in a single clock cycle. The fetch unit retrieves an instruction (I1) in the first clock cycle and saves it in buffer B1 at the conclusion of the clock cycle. The fetch process for instruction I2 is carried out by the instructions fetch unit during the second clock cycle. With buffer B1 at its disposal, the decode unit executes the operation dictated by instruction I1. Instruction I1 has been decoded and instruction I2 is accessible by the conclusion of the second clock cycle. The execution unit completes step execute of I1 in the third clock cycle, whereas

However, the risks that arise during pipelining will not be discussed in this work. As a result, pipeline safety precautions were also disregarded.

IV. Application

Pipelining has the benefits of faster reaction times and higher throughput. This processor is useful not only for conventional computing applications like workstations, desktops, and laptops, but also as a part of other devices like mobile phones, digital cameras, PDAs, home appliances, antilock brake systems for cars, and many more.

V. Simulation result

The pipelining is divided into four sub stages such as fetch, decode, execute, store. The simulation of each stage is done by using Quartus and Modelsim simulator and is shown bellow. The technology schematic for each stage is also shown.The Verilog HDL can be used to design the each stage of pipelining.

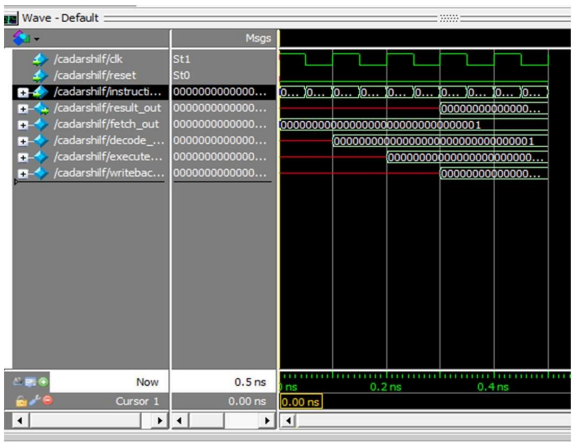

Fig. 4. Simulation 1

VI. RTL View

VII. Conclusion

In numerous systems, the pipelining principle offers numerous benefits. A portion of the risks are related to pipelining. The dangers notion is not discussed in this work. instruction

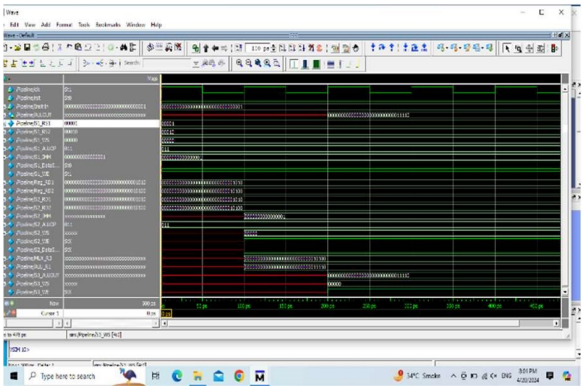

Fig. 5. Simulation 2

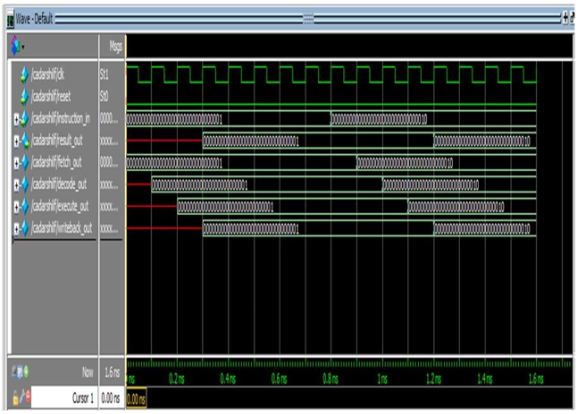

Fig. 6. Simulation 3

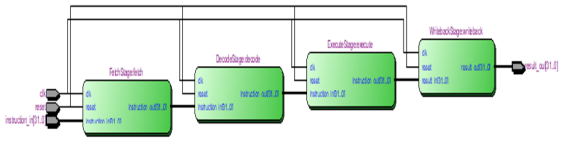

Fig. 7. RTL view

pipelining, which increases overall system throughput while lowering the CPI and accelerating operation or execution. The pipelining notion is one example of how a system's core idea can be significantly enhanced to increase system speed. In the future, when this idea is applied to different embedded systems, we will be able to see how it enhances performance.

REFERENCES

[1] Tannu Chhabra, Md Tauheed Khan "VLSI Design of a 16-bit Pipelined RISC Processor", International Journal of Electronics and Computer Science Engineering.

[2] J.L.Hennessy, D.A.Patterson.2003, Computer Organization and Design: The Hardware/Software Interface, 2nd Edition, Morgan Kaufmann.

[3] D. J. Smith. (2010), "HDL Chip Design, International Edition, Doone Publications.

[4] A.S. Tanenbaum. 2000, "Structured Computer Organization, 4th Edition, Prentice-Hall.

[5] Luker, Jarrod D., Prasad, Vinod B.2001, "RISC system design in an FPGA, MWSCAS 2001.

[6] Computer Organization Design. David A. Patterson and John L. Hennessy.