Design and Simulation of Four Stage Pipelining Architecture Using the Verilog

Rakesh M. R

M. Tech Student, Dept of ECE, Canara Engineering College, Mangalore, Karnataka, India

Abstract: ***The computer or any devices use the concept of parallelism for speedup of system operations. The one of parallelism technique is pipelining concept. Many devices using the pipelining for increase speed and throughput. The overall pipeline stage can be subdivided into stages such as fetch, decode, execute, store. In this paper the design and simulation of four stage pipeline can be done separately using the Xilinx ISE and Modelsim simulator. It shows how the each stage of pipeline performs the operations.***

Keyword: instruction, pipeline, speed of operation, processor cycle

1. Introduction

An instruction as the name instructs the computer what to do. In simple terms, every line of a program that we as users write instructs the computer to perform a series of operations. A computer understand these high level instructions by converting them into a machine understandable form known as machine language comprising of 1's and 0's. The work to be done in an instruction is broken into smaller steps (pieces), each of which takes a fraction of the time needed to complete the entire instruction. Each of these steps is a pipe stage (or a pipe segment). Pipe stages are connected to form a pipe.

An instruction cycle is the basic operation cycle of a computer. It is the process by which a computer retrieves a program instruction from its memory, determines what actions the instruction requires, and carries out those actions. This cycle is repeated continuously by the central processing unit (CPU), from boot up to when the computer is shut down. In simpler CPUs, the instruction cycle is executed sequentially: each instruction is completely processed before the next one is started. In most modern CPUs, the instruction cycle is instead executed concurrently in parallel, as an instruction pipeline: the next instruction starts being processed before the previous instruction is finished, which is possible because the cycle is broken up into separate steps.

Pipeling computer architecture has recieved considerable attention since the 1960s when the need for faster and more cost effective system became critical.the merit of pipeling is that it can help to match the speed of various subsystems without duplicating the cost of the entire system involved.as technology evolves faster and cheaper LSI circuits became available,and the future of pipeling either in a simple or complex form became more promising. Pipelining is a technique for overlapping operations during execution. Today this is a key feature that makes fast CPUs. Different types of pipeline are instruction pipeline, operation pipeline, and multi-issue pipelines.

In this paper we are going to simulate the pipelining stages such as fetch, decode, execute, store separately. The separate simulation can be done in order to show the working of every stage. The simulation procedure can be done by the use of Xilinx ISE tool and Modelsim simulator.

2. Concept of pipelining

Pipelining, a standard feature in RISC processors, allows for multiple instructions are overlapped in execution (to be executed during one clock cycle). In a pipelined processor, the datapath is divided into subsequent stages. Each stage is given an input from the previous stage, performs a specific operation on the given input, and passes the resulting output to the next stage in the datapath.

Each of these steps is called a pipe stage or a pipe segment. The stages are connected one to the next to form a pipe—instructions enter at one end, progress through the stages, and exit at the other end. The throughput of an instruction pipeline is determined by how often an instruction exits the pipeline. The time required between moving an instruction one step down the pipeline is a processor cycle. Because all stages proceed at the same time, the length of a processor cycle is determined by the time required for the slowest pipe stage, the longest step would determine the time between advancing pipe stage. In a computer, this processor cycle is usually one clock cycle (sometimes it is two, rarely more). Under these conditions, the speedup from pipelining equals the number of pipe stages. Thus, the time per instruction on the pipelined processor will not have its minimum possible value, yet it can be close.

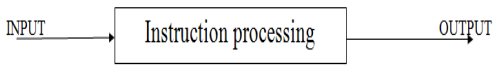


Figure1: Processing of instruction

Pipelining yields a reduction in the average execution time per instruction. The reduction can be viewed as decreasing the number of clock cycles per instruction (CPI), as decreasing the clock cycle time, or as a combination. If the starting point is a processor that takes multiple clock cycles per instruction, then pipelining is usually viewed as reducing the CPI. This is the primary view we will take. If the starting point is a processor that takes one (long) clock cycle per instruction, then pipelining decreases the clock cycle time.

The time required for moving an instruction from one stage to the next: a machine cycle (often this is one clock cycle). The execution of one instruction takes several machine cycles as it passes through the pipeline. After a certain time (N-1 cycles) all the N stages of the pipeline are working: the

pipeline is filled. Now, theoretically, the pipeline works providing maximal parallelism (N instructions are active simultaneously). Apparently a greater number of stages always provide better performance. However a greater number of stages increase the overhead in moving information between stages and synchronization between stages. With the number of stages the complexity of the CPU grows.

Pipelining is achieved: the sub-tasks into which instruction execution has been divided are instruction fetch, instruction decode, instruction execute, store result. Each of these sub-tasks, which is executed by a pipeline stage, produces intermediate results that must be stored before an instruction may move on to the next stage. By breaking up execution into smaller sub-tasks, it is possible to overlap the different sub-tasks of several different instructions simultaneously. If the intermediate results of the various sub-tasks are not stored, they would be lost: during the next cycle another instruction is taken for process. For instance, after an instruction is fetched, it is necessary to store the fetched instruction somewhere, because the output of the instruction memory will be different on the following cycle, the fetch stage will be fetching a completely different instruction.

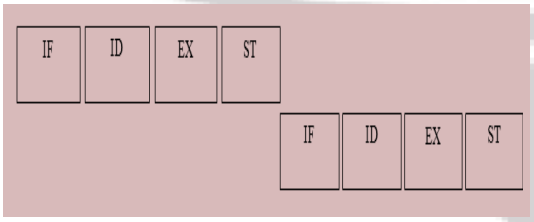


Figure 2: Unpipelined processor

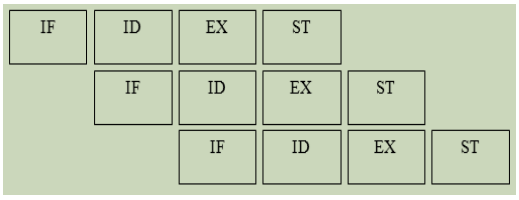


Figure 3: Pipelined processor

2.1 Performance of Pipelining

Pipelining increases the CPU instruction throughput, the number of instructions completed per unit of time, but it does not reduce the execution time of an individual instruction. In fact, it usually slightly increases the execution time of each instruction due to overhead in the control of the pipeline. The increase in instruction throughput means that a program runs faster and has lower total execution time, even though no single instruction runs faster. The fact that the execution time of each instruction does not decrease puts limits on the practical depth of a pipeline. In addition to limitations arising from pipeline latency, limits arise from imbalance among the pipe stages and from pipelining overhead. Imbalance among the pipe stages reduces performance since the clock can run no faster than the time needed for the slowest pipeline stage. For unpipelined processor the clock per instruction can be high. But by the use of the pipelining the CPI decrease and speed of execution increases.

Average instruction execution time=clock* average CPI
Speed from pipelining can be given by;

$$\frac{\text{Average instruction execution time unpipelined}}{\text{Average instruction execution time pipelined}}$$

2.2 Use of Pipeline

Suppose we execute 100 instructions. Single Cycle Machine: 45 ns/cycle x 1 CPI x 100 inst = 4500 ns. Multi cycle Machine: 10 ns/cycle x 4.6 CPI (due to inst mix) x 100 inst = 4600 ns. Ideal pipelined machine: 10 ns/cycle x (1 CPI x 100 inst + 4 cycle drain) = 1040 ns

2.3 Pipeline characteristics

Latency: The amount of time that a single operation takes to execute. The latency of pipelined processor determines how often dependent instruction may be executed.

Throughput: The rate at which operation get executed (generally expressed as operations/seconds or operations/cycles). Throughput > 1/latency, since instruction execution is overlapped.

CPI: Pipeline yields a reduction in cycles per instruction.

Processor cycle is time required between moving an instruction one step down the pipeline. 1 processor cycle = 1 CPU clock cycle. The slowest pipeline stage thus determines the length of the processor cycle. Pipelining aims to reduce the average execution time per instruction (CPI). Ideally: CPI pipeline = CPI / # of pipeline stages.

Potential speedup = Number pipe stages. An unbalanced length of pipe stages reduces speedup.

3. Phases of Pipeline

Instruction execution can be divided into four phases. These are;

- Instruction fetch (IF)
- Instruction decode & operand fetch (ID)
- Execution (EX)
- Store (ST)

We used a register transfer notation to describe what happens in each phase of instruction execution.

3.1 Instruction fetch (IF)

In this phase, the instruction to be executed next is fetched from memory using PC (program counter) as the address of the memory location (read the instruction from the memory). Corresponding location in the (instruction) memory is accessed and is assigned to the instruction register (IR).

IR <-- Mem [PC];

Next, the program counter is incremented to point to the next instruction. This is done by adding 1 to the current program counter. The new value of PC is stored in the NPC register.

NPC <-- PC + 1;

3.2 Instruction Decode & Operand Fetch (ID)

Once the Instruction Register is loaded with the instruction, decoding of that instruction along with operand fetching starts (decode the instruction and fetch the source operand(s)). Depending upon the instruction format (R, I or J), opcode along with the function code fields are used to decode the instruction. Simultaneously, the two source operands are fetched using the Rs1 and Rs2 field of the instruction. The fetched operands are stored in two internal register A and B. As operand fetch takes place even before the instruction decoding, fetching operand values into A and B registers take place for all three instruction forms (R, I and J) even though some instructions may have one or zero source operands. Simultaneous register fetch is possible due to the fixed field decoding (source operand fields are always at the position in all instruction formats). Thus,

A <-- Rs1; B <- Rs2;

3.3 Execution (EX)

In this phase, perform the operation specified by the instruction.

Arithmetic instructions are executed as follows. The operands (A and B register) for R-format and (A and Imm) for I-format are sent to the ALU with the appropriate control signal for performing the arithmetic operation. The control signals for the ALU and the MUX used to choose the second operand (between B and Imm) are derived by the control unit using the opcode and function code fields of the instruction in IR. Thus,

ALU_out <-- A (+) B (for instruction in R-format)
ALUo_ut <-- A (+) Imm (for instruction in I-format)

Data transfer instructions, only computing the effective address of the memory location is done in the EX phase. For example, in case of a LOAD instruction,

LD R2, 8(R1)

the decode and operand fetch phase has fetched the source register (R1) in A and the displace has been sign extended in Imm register.

Control instructions (branching (by PC relative address), the branch target address is calculated by adding the offset (given in the offset field of the instruction, loaded and sign extended in the Decode phase in the Imm register, to the Next Program Counter (NPC) value. That is,

ALU_out <-- NPC + Imm

For this, the first input operand is NPC which is Multiplexed with operand from the A register.

Store (ST) / (result write)

In this phase, for ALU and Load instructions, the result value is written in the destination register (store the result in the destination location). For branch instruction, PC is loaded with the branch target address (calculated in the EX phase) or NPC depending whether or not the condition specified in conditional branch instruction is true. For ALU instruction,

Rd <-- ALU_out

For LD instruction

Rd <-- LDR

The computer is controlled by a clock whose period is such that the fetch, decode, execute, store steps of any instruction can each be completed in one clock cycle. In the first clock cycle, the fetch unit fetches an instruction I1 and stores it in buffer B1 at the end of the clock cycle. In the second clock cycle, the instructions fetch unit proceeds with the fetch operation for instruction I2. The decode unit performs the operation specified by instruction I1, which is available to it in buffer B1. By the end of the second clock cycle, the decoding of instruction I1 is completed and instruction I2 is available. Step execute of I1 is performed by the execution unit during the third clock cycle, while instruction I3 is being fetched by the fetch unit.

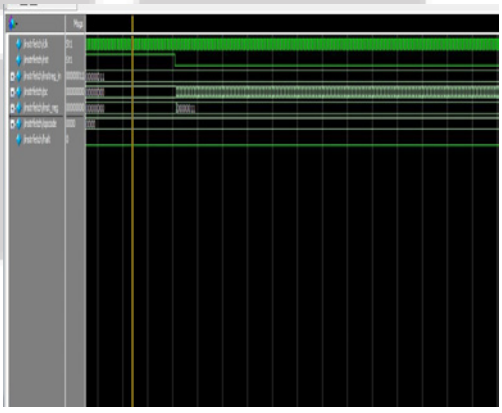
But in this paper we are going to ignore the hazards takes place during the pipelining. Therefore the methods for avoiding pipeline hazards also ignored.

4. Application

The advantages of a pipelining are an increase in throughput and reduced response time. This processor can be used not only in traditional computing applications such as desktops, laptops, workstation etc. but also as a component in another piece of technology such as either a cell phone, digital camera, portable digital assistant, household appliances, automobile antilock brake system and many more.

5. 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 Xilinx ISE 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.



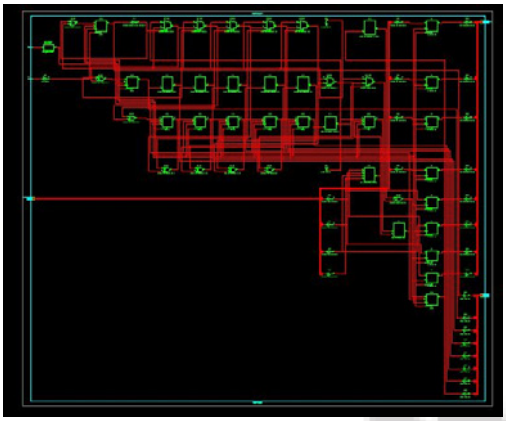
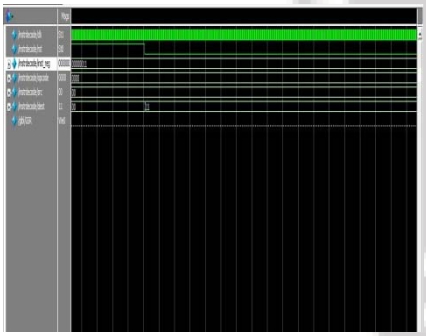


Figure 4:Waveform and Technology schematic for Fetch(IF) stage



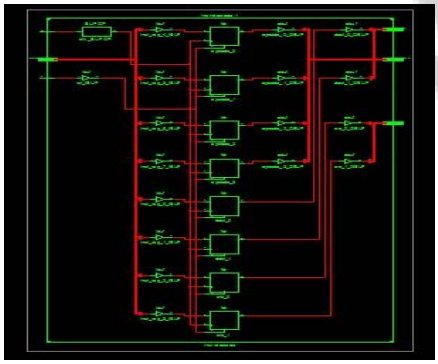
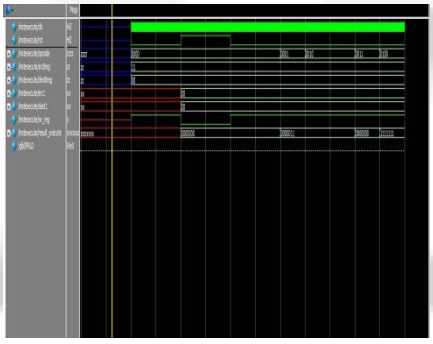


Figure 5:Waveform and Technology schematic for Decode(ID) stage



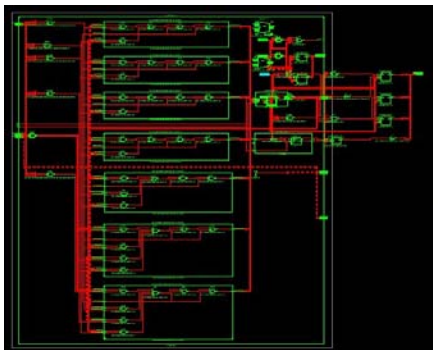
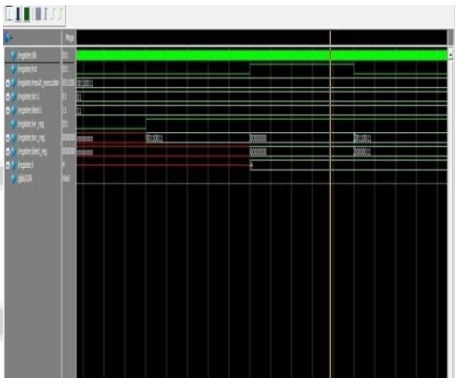


Figure 6:Waveform and Technology schematic for Execute(EX) stage



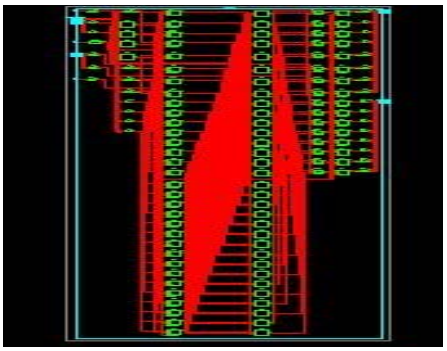


Figure 7:Waveform and Technology schematic for Store(ST) stage

6. Conclusion

The pipelining concept takes lot of advantages in many of the systems. The pipelining has some of the hazards. In this paper the hazards concept is ignored. The pipelining of instructions which reduce the CPI, increases the speed of execution or operation, and also increase throughput of overall system. This is basic concept of any system and lot of improvement can be done in pipelining concept to increase the speed of system. The future scope is to apply the concept to different embedded systems and we can see how the performance increases by this concept.

References

[1] Tannu Chhabra, Md Tauheed Khan "VLSI Design of a 16-bit Pipelined RISC Processor", International Journal of Electronics and Computer Science Engineering, ISSN 2277-1956/V1N3-1858-1861

[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, v2, , p532536.
[6] Computer Organization & Design. David A. Patterson and John L. Hennessy, ISBN 1-55860-428-6, p 476-501, 525-256.

Author Profile

Mr. Rakesh M. R received his B.E degree in Electronics and Communication from KVG College of Engineering Sullia in 2012. Currently he is pursuing M. Tech degree in Electronics at Canara Engineering College, Mangalore. His areas of interest are VLSI and Image Processing.