NOVATEUR PUBLICATIONS
INTERNATIONAL JOURNAL OF INNOVATIONS IN ENGINEERING RESEARCH AND TECHNOLOGY [IJIERT]
ISSN: 2394-3696
VOLUME 2, ISSUE 6 JUNE-2015

INCREASING THE THROUGHPUT USING EIGHT STAGE PIPELINING

Ms. Sobiya Ambreen
Department of Electronics and Communication VVIET, Mysuru, India.

ABSTRACT

Using re-programmable logic components along with HDL languages encompasses wider and wider areas of practical applications, becoming a standard of complex digital system design. One of the basic tasks, which are to be carried out in the process of design, is obtaining the highest efficiency of the solution under design. Thereby designers are still looking for methods making it possible to speed up design processing time. Pipelining mechanism is one of these methods. It helps to speed up some dedicated operations. In the early stage of design, a given unit described by high level language, is divided into some independent parts, which are synchronized with each other via intermediate registers and synchronization signal (pipelining mechanism). 8-stage pipelining is the key implementation technique used to make fast CPUs. It is an optimization technique used to speed up instruction execution. Throughput of an instruction pipeline is increased while latency is decreased for each instruction execution. This new 8-stage pipelining includes two instruction fetch, one instruction decode, two execution, two memory and one write back stages. It describes advantages of both speed and suitability for synthesizable RISC design.

INTRODUCTION

Pipelining is an implementation technique whereby multiple instructions are overlapped in execution; it takes advantage of parallelism that exists among the actions needed to execute an instruction. Today, pipelining is the key implementation technique used to make fast CPUs. A pipeline is like an assembly line. In an automobile assembly line, there are many steps, each contributing something to the construction of the car. Each step operates in parallel with the other steps, though on a different car. In a computer pipeline, each step in the pipeline completes a part of an instruction. Like the assembly line, different steps are completing different parts of different instructions in parallel. 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, just as cars would in an assembly line.

In an automobile assembly line, *throughput* is defined as the number of cars per hour and is determined by how often a completed car exits the assembly line. Likewise, the throughput of an instruction pipeline is determined by how often an instruction exits the pipeline. Because the pipe stages are hooked together, all the stages must be ready to proceed at the same time, just as we would require in an assembly line. 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, just as in an auto assembly line, the longest step would determine the time between advancing the line. In a computer, this processor cycle is usually one clock cycle (sometimes it is two, rarely more).

OBJECTIVE

Increasing the throughput and decreasing the latency per instruction.
The pipeline designer's goal is to balance the length of each pipeline stage, just as the designer of the assembly line tries to balance the time for each step in the process. If the stages are perfectly balanced, then the time per instruction on the pipelined processor assuming ideal conditions is equal to

$$\frac{\text{Time per instruction on unpipelined machine}}{\text{Number of pipe stages}}$$

Under these conditions, the speedup from pipelining equals the number of pipe stages, just as an assembly line with *n* stages can ideally produce cars *n* times as fast. Usually, however, the stages will not be perfectly balanced; furthermore, pipelining does involve some overhead. Thus, the time per instruction on the pipelined processor will not have its minimum possible value, yet it can be close. Pipelining yields a reduction in the average execution time per instruction. Depending on what you consider as the base line, 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 speed of execution of programs is influenced by many factors. One way to improve performance is to use faster circuit technology to build the processor and the main memory. Another possibility is to arrange the hardware so that more than one operation can be performed at the same time. In this way, the number of operations performed per second is increased even though the elapsed time needed to perform any one operation is not changed.

SCOPE OF WORK

Two real-time applications of pipelining technology are, graphics hardware for processing pixels is extremely latency tolerant not unusual to find pipelines that have 10's of stages[1]. Graphics pipelines are never flushed, high clock rate is extremely important because of large number of pixels (> 1 Million) that have to be supplied every screen, at >30 updates per second. Microprocessor instruction pipelines are not very latency tolerant, most CPU pipelines are only about 5-10 stages. Works and researches on pipelining always aim to increase throughput and decrease latency.

LITERATURE SURVEY

Pipelining increases the number of simultaneously executing instructions and the rate at which instructions are started and completed. Pipelining does not reduce the time it takes to complete an individual instruction, also called the latency. Pipelining improves instruction throughput rather than individual instruction execution time or latency [1].VHSIC hardware description language (VHDL) is defined, VHDL is a formal notation intended for use in all phases of the creation of electronic systems. Because it is both machine readable and human readable, it supports the development, verification, synthesis, and testing of hardware designs; the communication of hardware design data; and the maintenance, modification, and procurement of hardware. Its primary audiences are the implementers of tools supporting the language and the advanced users of the language [12], [13].Reduced Instruction Set Computer (RISC) architectures are the basis of modern high performance processors. They have a simplified instruction set, with register to register arithmetic instructions, memory can only be accessed by load and store instructions, usually with single and simple addressing mode; the instruction have the fixed size and few simple formats. RISC processors use pipelining and their instruction set allows tight control of their pipeline by the software. Optimizing compilers perform instruction scheduling so as to exploit the parallelism offered by the pipeline data path. RISC processors achieve high performance at low cost: the smaller and simpler circuits allow higher clock rates; the data path can use pipelining, and the compiler can exploit it. In addition, design time, design cost and silicon areas are reduced [14].

METHODOLOGY

8-stage pipelining describes the basic operation of the CPU pipeline, which includes descriptions of the delay instructions. The CPU has an eight-stage instruction pipeline; each stage takes one PCycle (one cycle of PClock, which runs at twice the frequency of Master Clock). Thus, the execution of each instruction takes at least eight PCycles (four Master Clock cycles)[16]. An instruction can take longer for example, if the required

data is not in the cache, the data must be retrieved from main memory. Once the pipeline has been filled, eight instructions are executed simultaneously. Figure 1 shows the eight stages of the instruction pipeline; the next section describes the pipeline stages.

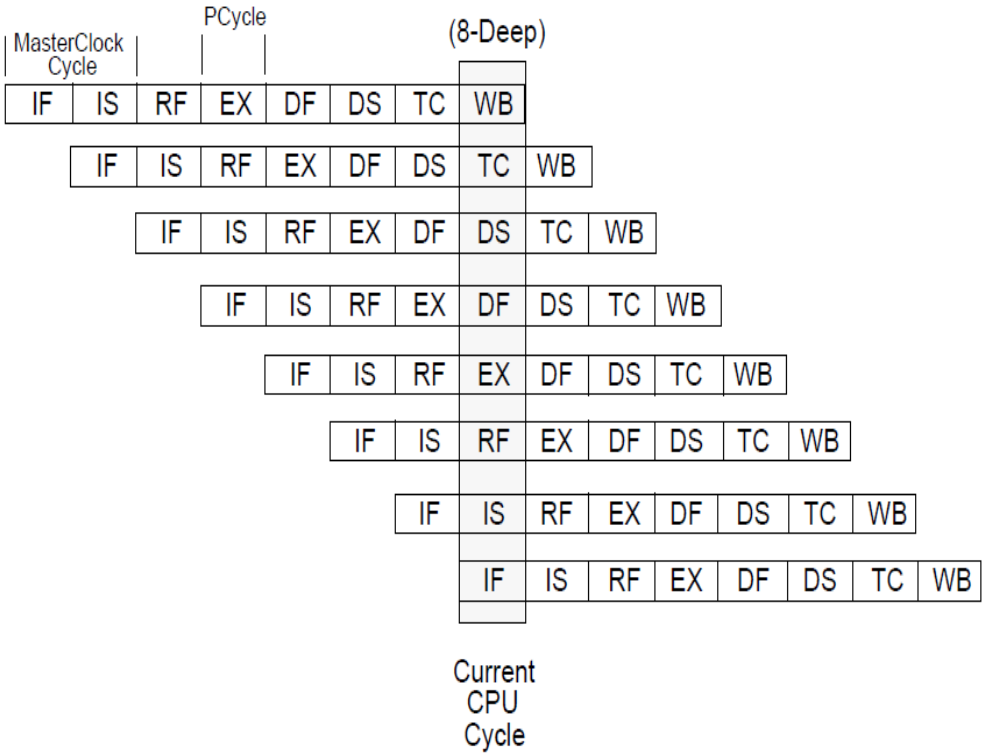

Figure 1: Instruction Pipeline Stages.

Pipeline stages

This section describes each of the eight pipeline stages:

IF - Instruction Fetch, First Half

IS - Instruction Fetch, Second Half

RF - Register Fetch

EX – Execution

DF - Data Fetch, First Half

DS - Data Fetch, Second Half

TC - Tag Check

WB - Write Back

*IF - Instruction Fetch, First Half***:** During the IF stage, the following occurs:

1. Branch logic selects an instruction address and the instruction cache fetch begins.
2. The instruction translation look aside buffer (ITLB) begins the virtual-to-physical address translation.

IS - Instruction Fetch, Second Half: During the IS stage, the instruction cache fetch and the virtual-to-physical address translation are completed as shown in figure 2.

RF - Register Fetch: During the RF stage, the following occurs:

1. The instruction decoder (IDEC) decodes the instruction and checks for interlock conditions.
2. The instruction cache tag is checked against the page frame number obtained from the ITLB.
3. Any required operands are fetched from the register file.

EX – Execution: During the EX stage, one of the following occurs:

1. The arithmetic logic unit (ALU) performs the arithmetic or logical operation for register-to-register instructions.
2. The ALU calculates the data virtual address for load and store instructions.
3. The ALU determines whether the branch condition is true and calculates the virtual branch target address for branch instructions.

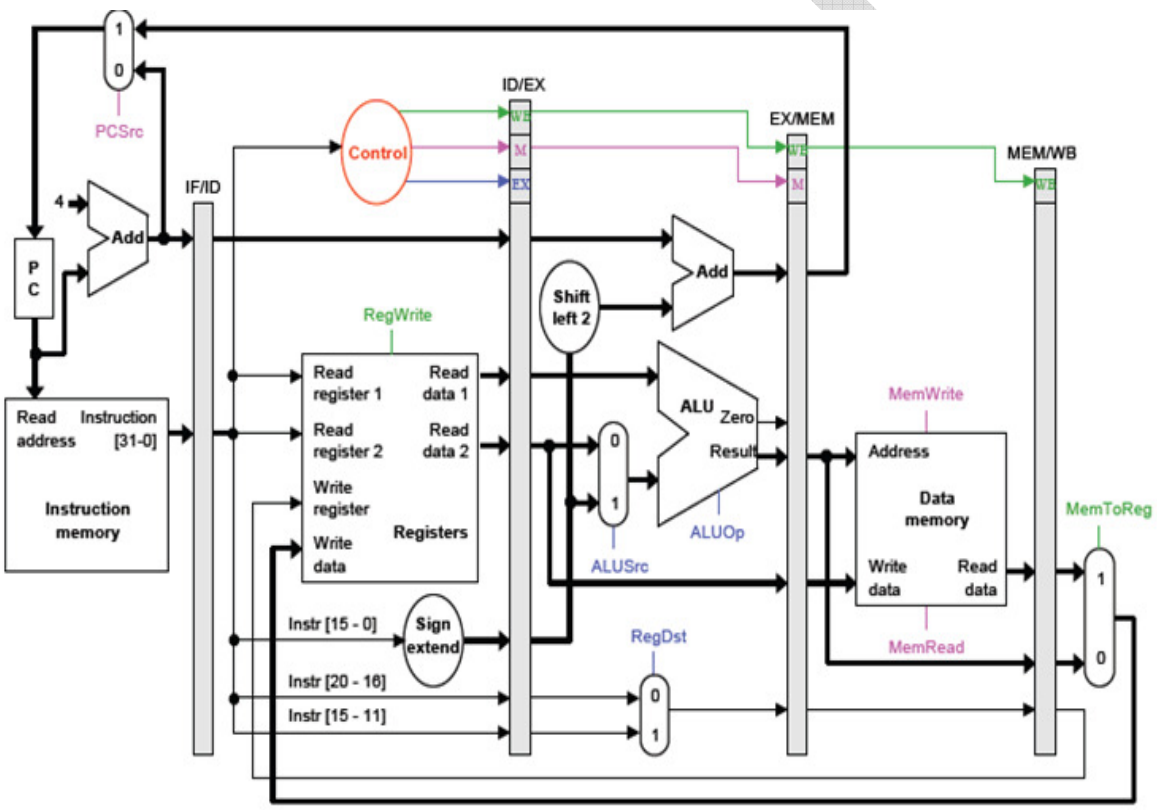

Figure 2: Block diagram of eight stage pipelining.

DF – Data Fetch, First Half: During the DF stage, one of the following occurs:

1. The data cache fetches and the data virtual-to-physical translation begins for load and store instructions.
2. The branch instruction address translation and translation look aside buffer (TLB) update begins for branch instructions [10].
3. No operations are performed during the DF, DS, and TC stages for register-to-register instructions.

DS - Data Fetch, Second Half: During the DS stage, one of the following occurs:

The data cache fetch and data virtual-to-physical translation are completed for load and store instructions. The Shifter aligns data to its word or double word boundary.

TC - Tag Check: For load and store instructions, the cache performs the tag check during the TC stage. The physical address from the TLB is checked against the cache tag to determine if there is a hit or a miss.

WB - Write Back: For register-to-register instructions, the instruction result is written back to the register file during the WB stage. Branch instructions perform no operation during this stage.

SOFTWARE REQUIREMENTS

1. Xilinx version 9.2.
2. VHDL language.

CONCLUSION

The work presented here describes a functional pipeline implementation design of a RISC processor designed using VHDL. Individual components of eight stage pipelining were designed using VHDL modules. The VHDL designs of the RISC processor were all simulated using Modelsim Simulator to ensure that the processors were functional, being simulated by the VHDL designs. The number of clock cycles per instructions is proportional to number of stages in pipelining. But the time required to execute an instruction is inversely proportional to number of pipelining stages. The goal was to increase the number of clock cycles and simultaneously decrease the execution time per instruction. As expected higher throughput and lower latency are successfully obtained. Higher the number of stages of pipelining complexity increases.

REFERENCES

[1] Patterson, D. A., Hennessy, J. L., "*Computer Organization and Design: The Hardware/Software Interface",* 2nd edition, Morgan Kaufmann Publishers, San Francisco, CA, 1998.
[2] *"Xilinx, XC4000 Series Field Programmable Gate Arrays Product Specification",* ver. 1.6, 1999.
[3] Altera, FLEX "*10K Embedded Programmable Logic Device Family Data Sheet",* ver. 4.2., 2003.
[4] Altera, "*MAX 7000 Programmable Logic Device Family Data Sheet",* ver. 6.02, 2002.
[5] MIPS Technologies, www.mips.com.
[6] SPIM, http://www.cs.wisc.edu/~larus/spim.html.
[7] Altera, "*MAX+PLUS II Getting Started Manual"*, ver. 6.0, 1995.
[8] Diab, H., Demashkieh, I., "*A reconfigurable microprocessor teaching tool*", IEEE Proceedings A, vol. 137, issue 5, September 1990.
[9] Gray, J., "*Designing a Simple FPGA-Optimized RISC CPU and System-on-a-Chip*", 2000.
[10] Altera, University Program UP2 Development Kit User Guide, ver. 3.0, 2003.

[11] MIPS Technologies, "*MIPS32™ Architecture For Programmers Volume I: Introduction to the MIPS32™ Architecture*", rev. 2.0, 2003.
[12] Brown, S., Vranesic, Z.,"*Fundamentals of Digital Logic with VHDL Design, McGraw-Hill Publishers"*, 2002.
[13] IEEE. "*IEEE Standard VHDL Language Reference Manual"*. IEEE, New York, NY, 2002. IEEE Standard 1076-2002.
[14] Land. B, "*Electrical Engineering 475 Microprocessor Architectures"*, http://instruct1.cit.cornell.edu/Courses/ee475/
[15] Takahashi, R., Ohiwa, H., "*Situated Learning on FPGA for Superscalar Microprocessor Design Education*", IEEE Proceedings of the 16th Symposium on Integrated Circuits and System Design, 2003.
[16] Altera, ByteBlaster MV Parallel Port Download Cable, ver. 3.3, 2002.
[17] Larus, J. R., "*SPIM S20: A MIPS R2000 Simulator*", 1993.