DESIGN AND IMPLEMENTATION OF HIGH PERFORMANCE 64-BIT VLIW MICROPROCESSOR

Thesis submitted in partial fulfillment of the requirements for the award of degree of

MASTER OF TECHNOLOGY

in

VLSI Design and CAD

Submitted By:

Manju Rani

Roll No. 601061014

Under the guidance of:

Ms. Harpreet Vohra

Assistant Professor

Department of Electronics and Communication Engineering

Thapar University

(Established under the section 3 of UGC Act, 1956)

PATIALA – 147004 (PUNJAB)

June 2012

CERTIFICATE

I hereby declare that the work which is being presented in the thesis entitled, **"Design and implementation of high performance 64-bit VLIW microprocessor"** in partial fulfillment of the requirement for the award of degree of Master of Engineering in VLSI Design & CAD submitted in Electronics and Communication Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of **Ms. Harpreet Vohra, Assistant Professor, ECED.**

The matter presented in this thesis has not been submitted in any other University/Institute for the award of degree.

Date: 02/07/2012

(Manju Rani)
Roll No: 601061014

It is certified that the above statement made by the student is correct to the best of my knowledge and belief.

(Ms. Harpreet Vohra)
Assistant Professor
ECED, Thapar University

Countersigned by:

Professor & Head
ECED, Thapar University
Patiala-147004

Dean of Academic Affairs
Thapar University
Patiala-147004

ACKNOWLEDGEMENT

To discover, analyze and to present something new is to venture on an untraded path towards and unexplored destination is an arduous adventure unless one gets a true torchbearer to show the way. I would have never succeeded in completing my task without the cooperation, encouragement and help provided to me by various people. Words are often too less to reveals one's deep regards. I take this opportunity to express my profound sense of gratitude and respect to all those who helped me through the duration of this thesis. I acknowledge with gratitude and humility my indebtedness to Ms. Harpreet Vohra, Assistant Professor, Electronics and Communication Engineering Department, Thapar University, Patiala, under whose guidance I had the privilege to complete this thesis. I wish to express my deep gratitude towards her for providing individual guidance and support throughout the thesis work.

I convey my sincere thanks to Head of the Department, Dr. Rajesh Khanna as well as PG Coordinator, Dr. Kulbir Singh, Associate Professor, Electronics and Communication Engineering Department, entire faculty and staff of Electronics and Communication Engineering Department for their encouragement and cooperation.

My greatest thanks are to all who wished me success especially my parents. Above all I render my gratitude to the Almighty who bestowed self-confidence, ability and strength in me to complete this work for not letting me down at the time of crisis and showing me the silver lining in the dark clouds. I do not find enough words with which I can express my feelings of thanks to my dear friends for their help, inspiration and moral support which went a long way in successful competition of the present study.

MANJU RANI

ABSTRACT

In the two decades since FPGAs introduced, the way which digital logic is designed and deployed has been radically changed. FPGAs have made possible entirely new types of applications. It is very important to design microprocessor as the part of core of electronic systems, so development and production. On making use of the technology of FPGA to design the microprocessor of logic function, it can quickly realize the function, complete design, cut down development cycle, save cost and quickly realize productions. The subject of the thesis is to design VLIW (is the abbreviation of "Very Long Instruction Word") microprocessor based on FPGA. It designs VLIW microprocessor which contains 64-bit instruction word and 192-bit data, each VLIW instruction word consists of three operations in parallel. The VLIW microprocessor can be designed using a pipeline technology of four stages, and have been implemented by taking advantage of the technology of FPGAs. According to the basic principle of VLIW microprocessor, it is rationally divided into five main modules: Fetch module, Decode module, Register file, Execute module, Writeback module. Each main module is reasonably divided again, and realized the function of every module based on the principle of FPGAs, so as to implement five main modules.

At last, the whole function of VLIW microprocessor is completely finished. It completes the data of empty operation, addition, subtraction, multiplication, load and move, read, comparison, XOR, NAND, NOR, NOT, shift left, shift right, barrel shift left, barrel shift right in VLIW microprocessor. It also solves the control competitions and the data competitions in execution module and register file in VLIW microprocessor by the technology of register's bypass. The concept, trait, principle and structure of the VLIW microprocessor are firstly introduced in the thesis. It presents the pipeline technology, introduces the basic design method and characteristics about FPGAs, elaborates the basic scheme of design a VLIW microprocessor and verifies 16 kinds of operational functions. Finally, it completes the design of VLIW microprocessor. The design is simulated on Modelsim SE and synthesized on Xilinx 13.1. The Thesis pays a significant attention to the analysis of VLIW in terms of pipelining, area so as to maximize throughput.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

ALU	Arithmetic logical unit
ARM	Acorn RISC Machine
CISC	Complex instruction set computing
CMOS	Complementary digital signal processing
CPU	Central processing unit
FPGA	Field programmable gate array
FLU	Floating point unit
HDL	Hardware description languages
IC	Integrates circuit
ILP	Instruction level parallelism
IEEE	Institute of electrical and electronics engineers
ISE	Integrated software environment
IBM	International business machine
IOB	Input output blocks
LSD	Least significant digit
LUT	Look-up table
NAN	Not a number
PAR	Place and route
RAM	Random access memory
RTL	Register transfer level
RISC	Reduced instruction set computing
VHDL	Very high speed integrated circuits HDL
VLSI	Very large scale integration
VLIW	Very long instruction word

CHAPTER -1

INTRODUCTION

1.1 Motivation

Currently, the IC fabrication technology has advanced enough to allow unprecedented implementations of computer architectures on a single chip. Also, the current rate of process advancement allows implementations to be improved at a rate that is satisfying for most of the markets these implementations serve. In particular, the vendors of general-purpose microprocessors are competing for sockets in desktop personal computers (including workstations) by pushing the envelopes of clock rate (raw operating speed) and parallel execution. The market for desktop microprocessors is proving to be extremely dynamic. In particular, the x86 market has surprised many observers by attaining performance levels and price/performance levels that many thought out of reach. To overcome this lower level of instruction-set inertia, all it takes is a sufficiently better set of implementation characteristics, particularly absolute performance and/or price-performance.

This lower level of instruction-set inertia gives the vendors of embedded microprocessors the freedom and initiative to seek out new instruction sets. The relative success of RISC microprocessors in the high-end of the embedded market is an example of innovation by microprocessor vendors that produced a benefit large enough to overcome the market's inertia. Because of advances in IC fabrication technology and advances in high-level language compiler technology, it now appears that microprocessor vendors are compelled by the potential benefits of another change in microprocessor instruction sets. As before, the embedded market is likely to be first to accept this change. The new direction in microprocessor architecture is toward VLIW (very long instruction word) instruction sets. VLIW architectures are characterized by instructions that each specify several independent operations. The demand for high speed processing has been increasing as a result of expanding computer and signal processing applications. Reducing the time delay and power consumption are very essential requirements for many applications. This is compared to RISC instructions that typically specify one operation and CISC instructions that typically specify several dependent operations. VLIW instructions are necessarily

longer than RISC or CISC instructions. The main advantage of VLIW processors is that complexity is moved from the hardware to the software, which means that the hardware can be smaller, cheaper, and require less power to operate.

1.2 Aim of the Thesis Report

The objective is to design a 64-bit VLIW Microprocessor supporting the following instruction set: addition, subtraction and multiplication. Second objective is to model the dynamic branch prediction in 4-stage 64-bit microprocessor to achieve better throughput. Figure 1.1 shows the complete implementation steps in designing a processor. The programming objective of the pipelining fall into the following categories:

1. **Accuracy:** The application produces that results that are close to the correct results.
2. **Performance:** The application produces the most efficient code possible.
3. **Latency:** The application produces a single output with in less time.
4. **Throughput:** The application produces more number of tasks that can be completed per unit time.
5. **Area:** The application produces less number of flip flops and slices.

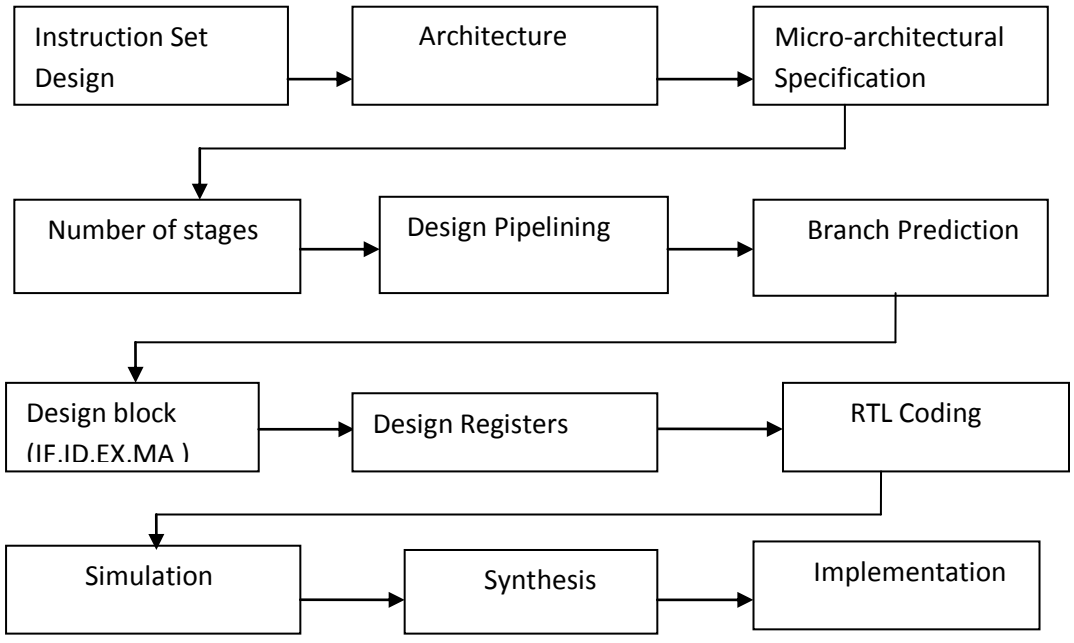


Figure 1.1: Implementation Steps

1.3 Organization of Thesis

This thesis consists of total seven chapters.

CHAPTER 1: INTRODUCTION - This chapter gives an introduction of the thesis report.

CHAPTER 2: VERY LONG INSTRUCTION WORD (VLIW) - In this chapter a discussion on the literature review of VLIW architectures is given.

CHAPTER 3: PIPELINING AND BRANCH PREDICTION MECHANISM - This chapter gives a brief introduction of pipelining. Pipelining and branch prediction mechanism.

CHAPTER 4: FIELD PROGRAMMABLE GATE ARRAY - This chapter is mainly concerned with FPGA and its design flow.

CHAPTER 5: PROPOSED ARCHITECTURE (VLIW) - This discusses about the proposed architecture (VLIW).

CHAPTER 6: SIMULATION AND SYNTHESIS RESULTS - shows the simulation and synthesis results for 64-bit VLIW processor.

CHAPTER 7: CONCLUSION - Derives the Conclusion and tells about future scope.

1.4 Tools Used

The tools used in the thesis are as follows:

Simulation Software:

1. Xilinx 13.1 and design compiler are used for synthesis and analysis.
2. Modelsim 10.1 has been used for modeling and simulation.

Hardware used:

Xilinx Spartan 3E (Family), XC4VFX12 (Device), Tool used HDL (Top Level Source Type), XST-VHDL/VERILOG (Synthesis Tool). ISE Simulator -VHDL/VERILOG (simulator) and Verilog (Preferred Language).

CHAPTER -2

VERY LONG INSTRUCTION WORD (VLIW)

2.1 Literature Review

Microprocessors have grown from 8 bits to 16 bits, 32 bits, and currently to 64 bits. Microprocessor architecture has also grown from complex instruction set computing (CISC) based to reduced instruction set computing (RISC) based on a combination of RISC-CISC based and currently very long instruction word (VLIW) based. This thesis discusses the hardware design and implementation of a 64-bit VLIW microprocessor capable of operating three operations per VLIW instruction word on FPGA technology. The first microprocessor was developed by Intel Corp in 1971. It was called 4004. The 4004 was a simple design compared to the microprocessors that we have today. However, back in 1971 the 4004 was a state-of-the-art microprocessor. Microprocessors today have grown manifold from their beginnings. Present-day microprocessors typically run in hundreds of megahertz ranging to gigahertz in their clock speeds. They have also grown from 8 bits to 16, 32, and 64 bits. The architecture of a microprocessor has also grown from CISC to RISC and VLIW.

Complex instruction set computing (CISC) is based on the concept of using as little instruction as possible in programming a microprocessor. CISC instruction sets are large with instructions ranging from basic to complex instructions. CISC microprocessors were widely used in the early days of microprocessor history. Reduced instruction set computing (RISC) microprocessors are very different from CISC microprocessors. RISC uses the concept of keeping the instruction set as simple as possible to allow the microprocessor's program to be written using only simple instructions. This idea was presented by John Cocke from IBM Research when he noticed that most complex instructions in the CISC instruction set were seldom used while the basic instructions were heavily utilized. Apart from the CISC and RISC microprocessors, there is a different generation of microprocessor based on a concept called very long instruction word (VLIW).

Table 2.1: Comparison of Different Architectures

Architectures	Re-Configurable Von-Neumann	Harvard	Modified Harvard	CISC		RISC		VLIW
Researchers	Rong-Jian & et. al. [3]	HU Yue-li & et. al. [4]	Cem Savas¸ & et. al. [5]	Y. Saito &et.al. [6]	J.P. Tual, & et.al.[7]	Seiji Miura, &et.al.[8]	Nick Richards & et. al. [9]	Weng Fook Lee,& et.al.[10]
Year	2006	2004	2009	1993	1995	2001	2003	2008
Frequency	100 MHz	26.9 - 34.7 MHz	80 MHz	40 MHz	66 MHz	100 MHz	520 MHz	152.25 MHz
Size	-	8-Bit	18-Bit	64-Bit	-	32-Bit	32-Bit	64 to 256 Bit
Power	27.74 mw	625.9 uw	-	6 W	5 W	5 W	5 W	5 W
Voltage	-	-	-	5v	5v	1.8v	1.8v	1.8v
Chip Size	15.6816 mm^2	0.485530 mm^2	-	16.3 mm x 12.7 mm	15mm x 15mm	0.28 mm x 0.28 mm	-	-
Technology	TSMC 0.18um	CSMC06 0.6um CMOS	-	0.8 um CMOS	0.5um CMOS	0.18um CMOS	0.18um CMOS HCMOS8D	0.18um CMOS
Methodology	FPGA	FPGA	FPGA	Full custom	Full custom	-	ASIC	FPGA & ASIC
Applications	Information security and file security	Higher Speed & low power applications and in SoC	Applications with special data types	Business Computer	Computer Applications	Used in Low Power systems	High-volume computer applications e.g. set up boxes	Sound and graphics processing & embedded applications

VLIW microprocessors make use of a concept of instruction level parallelism (ILP)-executing multiple instructions in parallel. VLIW microprocessors are not the only type of microprocessors that take advantage of executing multiple instructions in parallel. Superscalar super pipeline CISC/RISC microprocessors are also able to achieve parallel execution of instructions. [2].The term VLIW, and the concept of VLIW architecture itself, were invented by Josh Fisher in his research group at Yale University in the early 1980s. His original development of trace scheduling as a compilation technique for VLIW was developed when he was a graduate student at New York University. Prior to VLIW, the notion of prescheduling functional units and instruction-level parallelism in software was well established in the practice of developing horizontal microcode. Fisher's innovations were around developing a compiler that could target horizontal microcode from programs written in an ordinary programming language. He realized that to get good performance and target a wide-issue machine, it would be necessary to find parallelism beyond that generally within a basic block.

Fisher developed region scheduling techniques to identify parallelism beyond basic blocks. Trace scheduling is such a technique, and involves scheduling the most likely path of basic blocks first, inserting compensation code to deal with speculative motions, scheduling the second most likely trace, and so on, until the schedule is complete. Fisher's second innovation was the notion that the target CPU architecture should be designed to be a reasonable target for a compiler -the compiler and the architecture for VLIW must be co-designed. This was partly inspired by the difficulty Fisher observed at Yale of compiling for architectures like Floating Point systems' FPS164, which had a complex instruction set architecture (CISC) that separated instruction initiation from the instructions that saved the result, requiring very complicated scheduling algorithms.

Fisher developed a set of principles characterizing a proper VLIW design, such as self draining pipelines, wide multi-port register files, and memory architectures. These principles made it easier for compilers to write fast code.The first VLIW compiler was described in a Ph.D. thesis by John Ellis, supervised by Fisher. Multiflow produced the TRACE series of VLIW minisupercomputers, shipping their first machines in 1987. Multiflow's VLIW could issue 28 operations in parallel per instruction. The TRACE system was implemented in an MSI/LSI/VLSI mix packaged in cabinets, a technology that fell out of favor when it became more cost-effective to integrate all of the

components of a processor (excluding memory) on a single chip. Multiflow was too early to catch the following wave, when chip architectures began to allow multiple issue CPUs.

2.2 Introduction and Comparison between CISC, RISC, and VLIW Architecture

From the larger perspective, RISC, CISC, and VLIW architectures have more similarities than differences. The x86 processors belong to the CISC (Complex Instruction Set Computing) category, mainly because their instructions may be quite complex or have variable length. They use a relatively small number of registers, and are capable of accessing memory locations directly. Complex instructions are sequenced in microcode in modern CISC processors. A different line, derived from CISC, is represented by the RISC (Reduced Instruction Set Computing) processors introduced in the 1980s, which are characterized mostly by how they differ from CISC processors. The instructions are of fixed length, and of a regular format. Operations are performed on registers only, of which a larger number are available than on CISC processors.

The only memory operations are load and store. The hardware in RISC processors is simpler in principle than in CISC ones, because a RISC architecture relies more on the compiler for sequencing complex operations. Advances in both CISC and RISC processors were possible on one hand through technological progress in the manufacturing area, and on the other through new processor architecture and micro architecture features. a processor must reconstruct the implicit parallelism in the code from the serialized instruction stream, which constitutes a complicated process. To overcome the difficulty of finding parallelism in machine-level object code, other processor architectures were developed. Among these, VLIW (Very Long Instruction Word) has a special significance because of its legacy. In a VLIW processor, multiple instructions are packed together and issued in parallel to an equal number of execution units.

Table 2.2: Comparison between CISC, RISC, VLIW Architecture [11]

ARCHITECTURES CHARACTERISTIS	CISC	RISC	VLIW
INSTRUCTION SIZE	Varies	One size, usually 32 bits	One size
INSTRUCTION FORMAT	Field placement varies	Regular, consistent placement of fields	Regular, consistent placement of fields
INSTRUCTION SEMANTICS	Varies from simple to complex; possibly many dependent operations per instruction	Almost always one simple operations	Many simple, independent Operations
REGISTERS	Few, sometimes special	Many, general-purpose	Many, general-purpose
MEMORY REFERENCES	Bundled with operations in many different types of Instructions	Not bundled with operations, i.e., load/store architecture	Not bundled with operations, i.e., load/store architecture
HARDWARE DESIGN FOCUS	Exploit microcode Implementations	Exploit implementations One pipeline &no microcode	No complex design logic

2.3 VLIW (Very Long Instruction Word)

VLIW has been developed to exploit Instruction-Level Parallelism by using a long instruction word which contains multiple fixed numbers of operations. Those operations can be fetched, decoded, issued, and executed at the same time without causing any data or control hazards. Therefore, all operations within a single VLIW instruction must be absolutely independent. Very long instruction word (VLIW) describes a computer processing architecture in which a language compiler or pre-processor breaks program instruction down into basic operations that can be performed by the processor in parallel (that is, at the same time). These operations are put into a very long instruction word

which the processor can then take apart without further analysis, handing each operation to an appropriate functional unit. VLIW is sometimes viewed as the next step beyond the reduced instruction set computing (RISC) architecture, which also works with a limited set of relatively basic instructions and can usually execute more than one instruction at a time (a characteristic referred to as superscalar).

The main advantage of VLIW processors is that complexity is moved from the hardware to the software, which means that the hardware can be smaller, cheaper, and require less power to operate. The Crusoe family of processors from Transmeta uses very long instruction words that are assembled by a pre-processor that is located in a flash memory chip. Because the processor does not need to have the ability to discover and schedule parallel operations, the processor contains only about a fourth of the transistor s of a regular processor. The lower power requirement enables computers based on Crusoe technology to be operated by battery almost all day without a recharge. The Crusoe processors emulate Intel's x86 processor instruction set. Theoretically, pre-processors could be designed to emulate other processor architectures. Crusoe- Crusoe is a family of "smart" microprocessors from Transmeta that combines a relatively simple, low-powered hardware processor with software that makes the hardware processor look like an x86 Intel.

2.4 Current Commercially Available Superscalar/VLIW Microprocessors and DSPs

Several recent commercially-available general purpose processors and digital signal processors (DSPs) have been designed for the demanding computing needs in processing digital video, images, graphics and audio to enable new consumer-level applications [12]. These new processors employ instruction-level parallelism which includes the superscalar and very long instruction word (VLIW) computer architectures. Instruction-level parallelism allows for multiple CPU operations to be initiated in a single clock cycle. This is done by having multiple execution units' on-chip and/or by partitioning the ALU of a particular execution unit into multiple independent pieces (e.g., a 32-bit ALU split into two 16-bit ALUs).

Table 2.3: Lists Various Characteristics of these Advanced Microprocessors [13, 14]

Processor	Current clock frequency	Num. of operations issued per cycle	Availability of partitioned addition	Num. of multiplies issued per cycle	Die size	Process technology
Intel Pentium Pro	200 MHz	3	NO	1	195 mm^2	0.35 µm
Motorola PowerPC 604	150 MHz	4	NO	1	196 mm^2	0.5 µm
HP PA RISC 8000	200 MHz	4	Yes	2	345 mm^2	0.5 µm
Sun UltraSPARC I	200 MHz	4	Yes	2	315 mm^2	0.5 µm
TI TMS320C80	50 MHz	18	Yes	5	342 mm^2	0.6 µm

Intel Pentium Pro, Hewlett-Packard PA-RISC 8000, Sun Microsystems UltraSPARC, and Texas Instruments TMS320C80 are all recently-developed superscalar/VLIW processors designed to meet the increasing computing requirement due to the proliferation of multimedia. Table 2.3 lists various characteristics of these advanced microprocessors. Several new courses have been developed to cover such architectures and processors [13, 14], emphasizing more on the hardware aspects. However, we believe that real-time implementations require a systems approach including software, hardware, and integration. Thus, we have taken a balanced approach in our course that the programmers must understand the underlying hardware which executes their software.

2.5 Concept and Benefits (VLIW)

In VLIW architecture, parallel execution of multiple instructions is made possible by issuing a long instruction word. A single long instruction word is designed to achieve simultaneous execution of a fixed number of multiple operations. Those operations must

be independent of each other to avoid possible data hazards. Indeed, several independent instructions are integrated inside a very long instruction word.

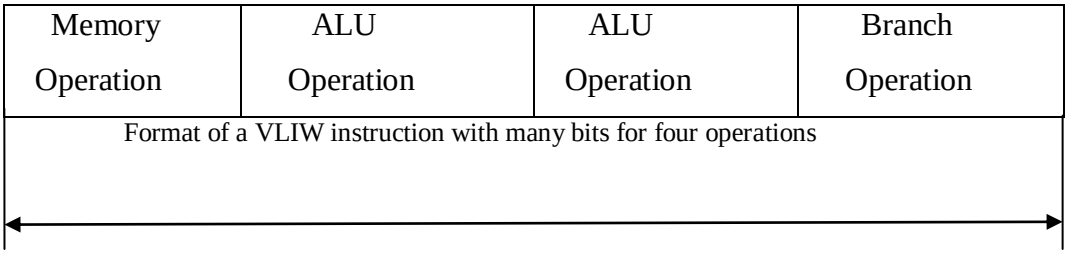


Figure 2.1: An Example of VLIW Instructions [16]

The VLIW instruction is wide enough to allow the concurrent operation of multiple functional units. Its size normally ranges from 64 to 128 bits, and even up to 1024 bits. Figure 2.1 shows a typical format of VLIW instructions. Many bits on the long instruction enable a single instruction word to sufficiently control the several functional units directly and independently in every cycle. Since it is the long instruction word which delivers the potential ILP to the hardware, a VLIW processor can be designed with a simpler hardware compared to an equivalent superscalar processor: it need not include the special units the run-time dependency check and instruction scheduling.

The block diagram of a simple VLIW processor is shown in Figure 2.2. VLIW architecture is, by essence, meant to activate multiple functional units at the same time. Therefore, the VLIW compiler should uncover independent operations to be executed in parallel. This means that the compiler must perform a detailed analysis on the data-flow and control-flow at compile time (which is when the potential ILP is fixed). Since the ILP within a basic block is quite limited, the VLIW architecture needs to examine more instructions to find more ILP. It is possibly achieved by looking at the instruction stream beyond the control-flow limits. For that purpose, several techniques such as loop unrolling and trace scheduling have been introduced in the VLIW design techniques. In addition, VLIW can uncover more parallelism by searching over a wider range of static code. Also, it is quite beneficial to know the source code structure to find parallelism in the VLIW architecture. However, several limitations such as long compilation time, not enough compatibility, and code explosion make VLIW architectures difficult to use in practice [15]. In conclusion, we can say that VLIW architectures do not have the hardware complexity of current superscalar architectures. The two approaches in the current high-performance microprocessor design.

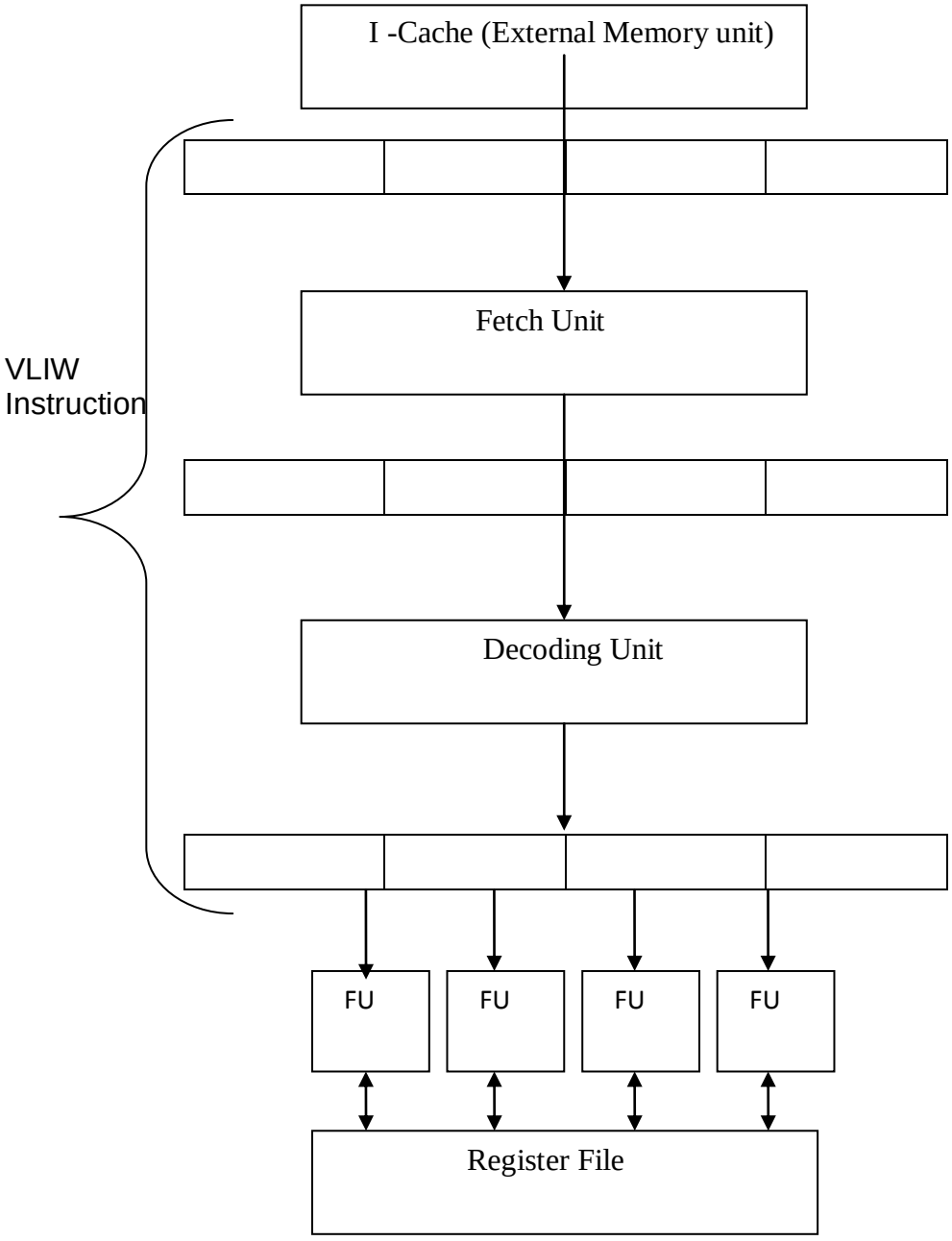


Figure 2.2: VLIW Hardware [16]

In the following Part, more detailed techniques for the VLIW architecture is explained. The key to higher performance in microprocessors for a broad range of applications is the ability to exploit fine-grain, instruction-level parallelism. [16]. some methods for exploiting fine-grain parallelism include:

- pipelining
- multiple processors
- superscalar implementation
- specifying multiple independent operations per instruction

Pipelining is now universally implemented in high-performance processors. Little more can be gained by improving the implementation of a single pipeline. Using multiple processors improves performance for only a restricted set of applications. Superscalar implementations can improve performance for all types of applications. Superscalar (super: beyond; scalar: one dimensional) means the ability to fetch, issue to execution units, and complete more than one instruction at a time. Superscalar implementations are required when architectural compatibility must be preserved, and they will be used for entrenched architectures with legacy software, such as the x86 architecture that dominates the desktop computer market.

Specifying multiple operations per instruction creates a very-long instruction word architecture or VLIW. A VLIW implementation has capabilities very similar to those of a superscalar processor-issuing and completing more than one operation at a time-with one important exception: the VLIW hardware is not responsible for discovering opportunities to execute multiple operations concurrently. For the VLIW implementation, the long instruction word already encodes the concurrent operations. [17].This explicit encoding leads to dramatically reduced hardware complexity compared to a high-degree superscalar implementation of a RISC or CISC. The big advantage of VLIW, then, is that a highly concurrent (parallel) implementation is much simpler and cheaper to build than equivalently concurrent RISC or CISC chips. VLIW is a simpler way to build a superscalar microprocessor.

CHAPTER -3

PIPELINING AND BRANCH PREDICTION MECHANISM

3.1 Introduction

It is a technique that allows for simultaneous execution of parts, or stages, of instructions to more efficiently process them. It is first introduced in IBM 7030 (Stretch Computer). 1986 was the first pipelined CISC processor. RISC processors in 80s were pipelined and were efforts to get IPC of 1. With a RISC processor, 1 instruction is executed while the next is being decoded and its operands are being loaded while the following instruction is being fetched all at the same time. shows the three, four and five stage VLIW Pipeline respectively.

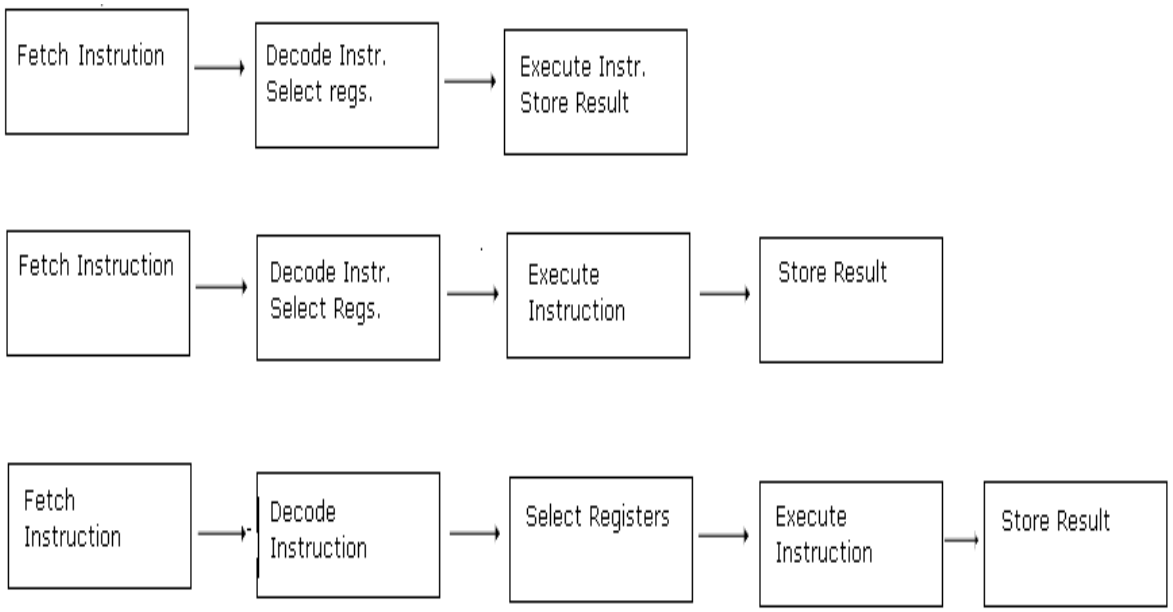


Figure 3.1: Three, Four and Five - stage VLIW Pipelines

Thus typical pipeline generally consists of four stages:

- Stage 1: Fetches instruction from memory.
- Stage 2: Decodes instruction and fetches any required operands
- Stage 3: Executes instructions
- Stage 4: Stores results

Each stage processes instructions simultaneously after a delay to fill the pipeline and this

allows CPU to execute 1 instruction per clock cycle.

Apart from the CISC and RISC microprocessors, there has a different generation of microprocessor based on a concept called very long instruction word (VLIW). VLIW microprocessors make use of a concept of instruction level parallelism (ILP) executing multiple instructions in parallel. [18]. VLIW microprocessors have not the only type of microprocessors that take advantage of executing multiple instructions in parallel. Superscalar superpipeline CISC/RISC microprocessors are also able to achieve parallel execution of instructions.

3.2 Principle of Pipelining

The basic principle behind pipelining is to allow to start the process of executing one instruction before the previous one has completed and it shows that even if there are delays in any one stage of the process for one instruction, it is still more efficient than non-pipelined processors. Figure 3.2 shows the processing of a sequence of instructions using a basic pipeline and Figure 3.3 shows the processing of a sequence of instructions using 4-stage pipelined.

1) Fetch
2) Decode
3) Execute
4) Writeback

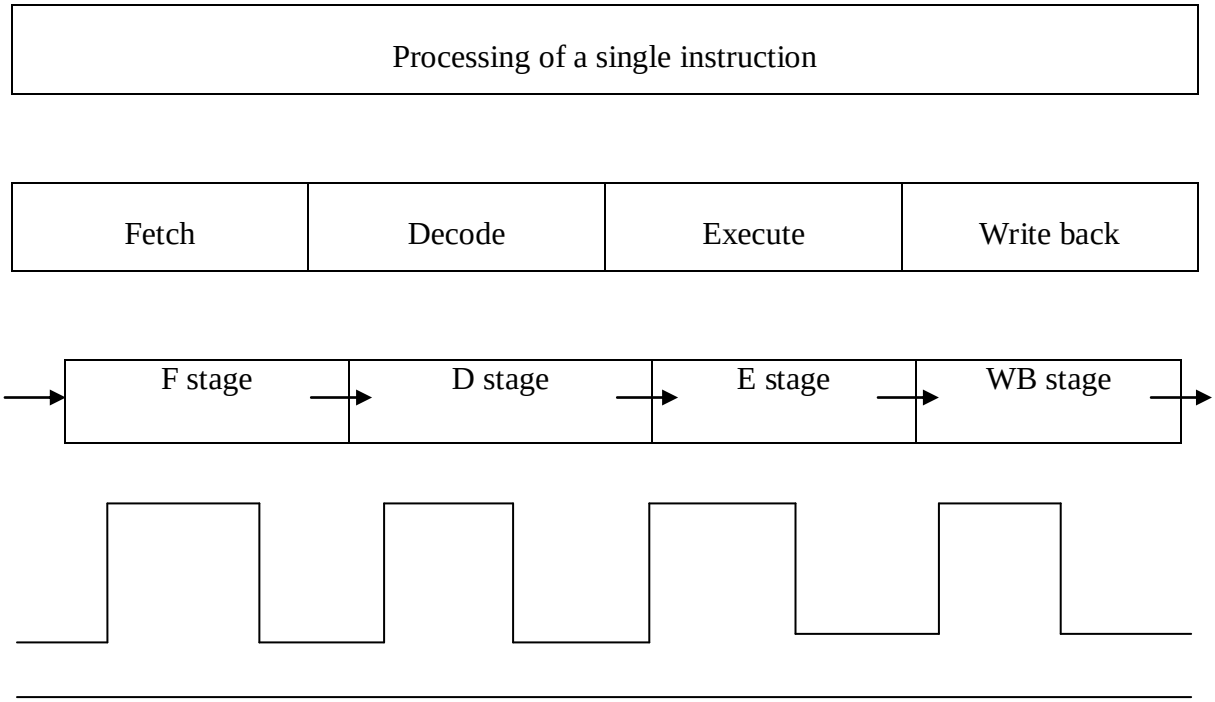


Figure 3.2: Processing of a sequence of instructions using basic pipeline [19]

F stage	D stage	E stage	WB stage

Cycle	In	In Processing				Out Finished
1 Cycle	Instr. 1	F 1				
2 Cycle	Instr. 2	F 2	D 1			
3 Cycle	Instr. 3	F 3	D 2	E 1		
4 Cycle	Instr. 4	F 4	D 3	E 2	WB 1	Instr. 1
5 Cycle	Instr. 5	F 5	D 4	E 3	WB 2	Instr. 2

Figure 3.3: Processing of a sequence of instructions using a 4-stage pipeline. [19]

3.3 Design Issues

Following are the design issues:

- Data dependencies and branch instructions have to be handled carefully. Data dependency means next instruction depends on result of last one which has not taken place due to previous one in pipeline. Conditional jumps may be problem if last stage in pipeline and condition changes later after jump has been processed

Design issues – single cycle instruction

- Microprocessor stalled when instruction stage does not take one clock cycle.
- Stalling can be because of delays in reading from memory, poor instruction set design, dependencies between instructions.

Performance issues in Pipelined systems

- Memory speed - caches. Fast memory b/w processor and slower memory.
- Copy from main memory also kept in cache to speed up further references.
- Caches - problem of coherency. Results kept in cache must go to main before it is Read or deleted in cache.
- Instruction Latency: Poor instructions that may take more than one clock should be avoided.
- Highly encoded instructions that use complex decoders
- Variable length instructions with multiple references to memory.
- Instructions that access main memory.

- Complex instructions that require multiple clocks like floating point multiplication.
- Dependency Issues – If one instruction sets the conditions in the condition code Register and next tries to read those bits, 2^{nd} has to wait for 1^{st} to complete
- Instruction scheduling – and common sub expression elimination.

3.4 Advantages of Instruction Pipeline

The main advantages of instruction pipeline are:

- It's possible to use several control units for processing instructions but a single Pipelined control unit offers some advantages
- Reduces requirement of hardware pipeline.
- Each stage performs only a portion of the pipeline stages and that no stage needs to incorporate a complete hardware control unit.
- Each stage only needs the hardware associated with its specific task.
- The instruction-fetch stage only needs to read an instruction from memory. This stage does not need the hardware used to decode or execute instructions
- Similarly a stage that decodes instructions does not access memory; the instruction fetch stage has already loaded the instruction from memory into an instruction register for use at the decode stage.
- Another advantage of instruction pipelines is the reduced complexity of the memory Interface.
- If each stage had a complete control unit, ALL stages can access the memory and it can cause memory access conflicts in which the CPU must take care of.
- In general practice, RISC CPU has their memory partitioned into instruction and data modules.
- The instruction-fetch stage reads data only from the instruction memory module at all other times; it is dealing with data in registers.
- The execute-instruction stage only access memory when it is reading data from the Data memory module. Custom designed memory can be configured to allow Simultaneous read/write access to different locations while avoiding memory access Conflicts by the execute-instruction and store-results stages of the pipeline.

3.5 Pipeline Clock Rate

The clock rate of the pipeline and the CPU is limited to its slowest stage.

Example 1: 4 stage pipeline with delays of 20ns, 20ns, 100ns, 40ns.

The clock period must be at least 100ns to handle the delay at the 3rd stage (100ns). This Results in a maximum clock rate of 10MHz.

Thus, when all stages have same delay time, the pipeline will achieve maximum Performance.

The Speedup ratio (Sn) is expressed by this formula:

Sn = n * T1 / (n + k – 1) * Tk

n = number of instructions

T1 = time needed to process 1 instruction (non-pipeline)

k = number of stages in the pipeline

Tk = clock period of the pipeline

Example 2: Let T1 = 180 ns (time needed to process 1 instruction)

k = 4 (stages in the pipeline)

Tk = 50 ns (clock period of the pipeline)

Applying the formula, it results out as:

Sn = n * 180 / (n + 4 – 1) * 50

For steady state ($n \rightarrow \infty$), the maximum speedup is Sn = 180 / 50 is 3.6.

But in reality, the speedup would be slightly less than this for some reasons.

- The reason is that this does not account for the first few cycles needed to fill the pipeline; in addition the 180ns includes the time needed for the latches at the end of each stage. In a non-pipelined CPU, these latches and their associated delays do Not exist and the actual time needed to process an instruction would be slightly less than 180ns.
- The other reason is pipelines do cause and encounter problems.
    - One problem is memory access. A pipeline must fetch an instruction from memory in a single clock cycle but main memory usually is not fast enough to supply data quickly. RISC processors must include cache memory which is at least as fast as the pipeline. If the RISC CPU cannot fetch 1 instruction per clock cycle, it cannot execute 1 instruction per clock

cycle. The cache must separate instructions and data to prevent memory conflicts from different stages of the pipeline.

- Another problem is caused by branch statements. When a branch statement is executed, the instructions already in the pipeline should not exist. They are instructions located sequentially after the branch statement and not the instructions to which the branch statement jumps. There is not much that the pipeline can do about this but instead an optimizing compiler is needed to reorder the instructions to avoid this problem.

- **Example 3:** Consider a non pipelined machine with 6 execution stages of lengths 50ns, 50ns, 60ns, 60ns, 50ns, and 50ns as shown in figure 3.4.

  a) Find the instruction latency on this machine.

  b) How much time does it take to execute 100 instructions?

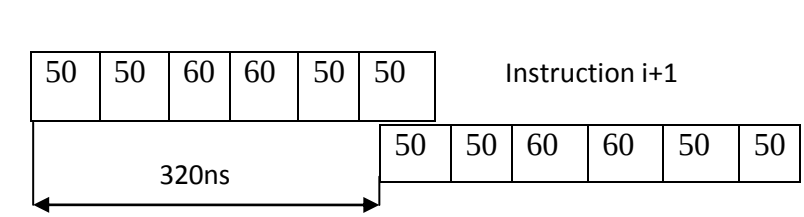


Figure 3.4: Non-Pipelined Machine

- Instruction latency = 50+50+60+60+50+50= 320 ns
- Time to execute 100 instructions = 100*320 = 32000 ns

- Example 4: Suppose we introduce pipelining on this machine as shown in figure 3.5. Assume that when introducing pipelining, the clock skew adds 5ns overhead to each execution stage.

  a) What is the instruction latency on the pipelined machine?

  b) How much time does it take to execute 100 instructions?

  Solution: In the pipelined implementation, the length of the pipe stages must all be the same that is the speed of the slowest stage plus overhead.

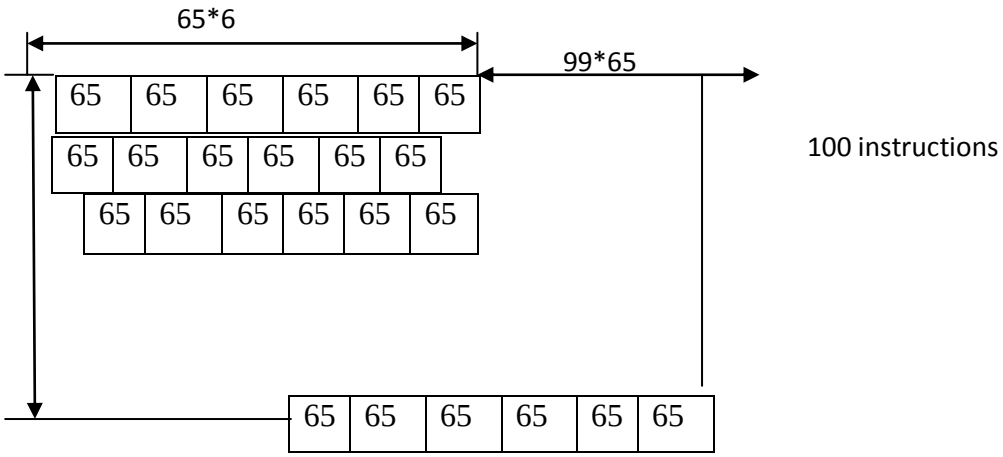


Figure 3.5: Pipelined Machine

- With 5ns overhead it comes to:

  The length of pipelined stage = MAX (lengths of non-pipelined stages) + overhead

  = 60 + 5 = 65 ns

- Instruction latency = 6x65 ns =390ns
- Time to execute 100 instructions = 65*6*1 + 65*1*99 = 390 + 6435 = 6825 ns

  What is the speedup obtained from pipelining?

  Solution: Speedup is the ratio of the average instruction time without pipelining to the average instruction time with pipelining.

  Average instruction time not pipelined = 320 ns

  Average instruction time pipelined = 65 ns

  Speedup = 320 / 65 = 4.92

Thus,

- Pipelining doesn't help latency of single task, it helps throughput of entire workload
- Pipeline rate limited by slowest pipeline stage
- Multiple tasks operating simultaneously
- Potential speedup = Number of pipe stages
- Unbalanced lengths of pipe stages reduces speedup
- Time to "fill" pipeline and time to "drain" it reduces speedup

3.6 Pipeline Hazards

Pipeline hazards are the conditions that lead to incorrect behavior if not fixed. The performance gain from using pipelining occurs because we can start the execution of a new instruction each clock cycle. In a real implementation this is not always possible.

Another important thing is that in a pipelined processor, a particular instruction still takes at least as long to execute as non-pipelined. Thus, Pipeline hazards prevent the execution of the next instruction during the appropriate clock cycle. Hazard will cause a pipeline stall.

• Some performance expressions involving a realistic pipeline in terms of CPI. It is assumed

That the clock period is the same for pipelined and unpipelined implementations.

Speedup = CPI Unpipelined / CPI pipelined

= Pipeline Depth / (1 + Stalls per Instruction)

= Average Instruction Time Unpipelined / Average Instruction Time pipelined

Thus, the speedup equation for pipelining is:

CPI pipelined = Ideal CPI + Pipeline stall clock cycles per instruction

$$\text{Speedup} = \frac{\text{Ideal CPI} \times \text{Pipeline depth}}{\text{Ideal CPI} + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle}_{\text{unpipelined}}}{\text{Clock Cycle}_{\text{pipelined}}}$$

$$\text{Speedup} = \frac{\text{Pipeline depth}}{1 + \text{Pipeline stall CPI}} \times \frac{\text{Clock Cycle}_{\text{unpipelined}}}{\text{Clock Cycle}_{\text{pipelined}}}$$

3.6.1 Types of Hazards

There are three types of hazards in a pipeline, they are as follows:

1. Structural Hazards: are created when the data path hardware in the pipeline cannot support all of the overlapped instructions in the pipeline and when two different instructions use same hardware in same cycle.
2. Data Hazards: When there is an instruction in the pipeline that affects the result of another instruction in the pipeline and when two instructions use the same storage.
3. Control Hazards: The PC causes these due to the pipelining of branches and other instructions that change the PC.

3.6.1.1 Structure Hazards

- Structural hazards result from the CPU data path not having resources to service all the required overlapping resources.
- Suppose a processor can only read and write from the registers in one clock cycle. This would cause a problem during the ID and WB stages.
- Assume that there are not separate instruction and data caches, and only one memory access can occur during one clock cycle. A hazard would be caused during the IF and MEM cycles. Figure 3.6 shows the structure hazards.

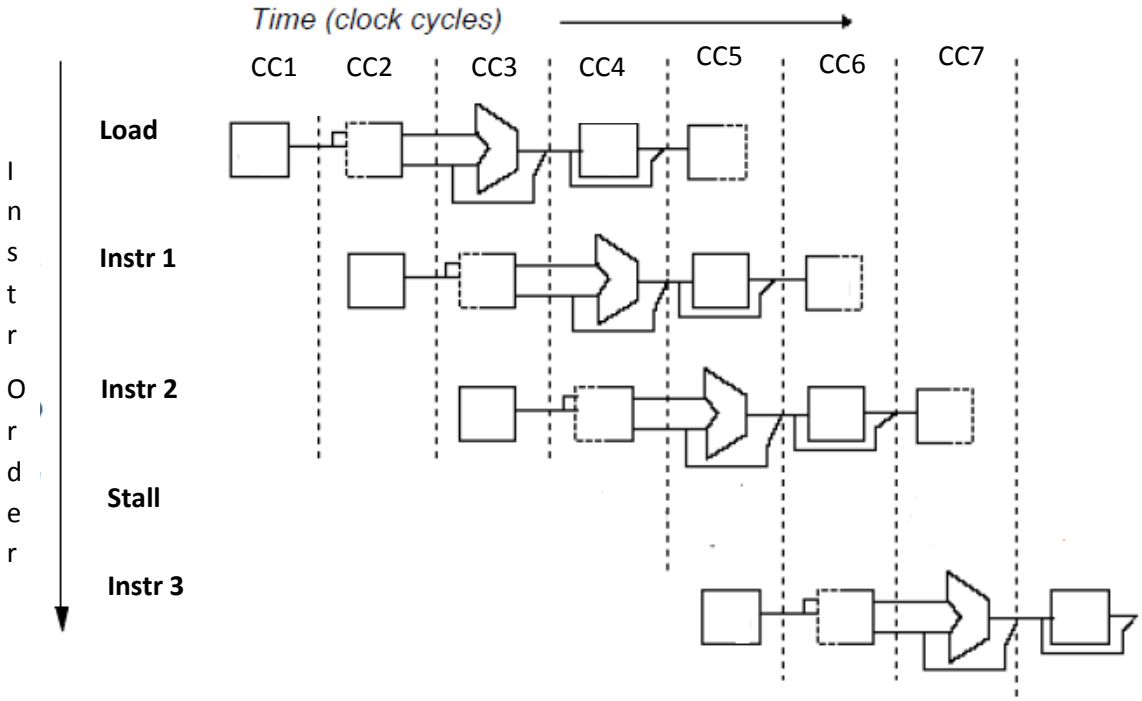


Figure 3.6: Structure Hazards [19]

- **Dealing with Structure hazards**

1. Stall

- Low cost, simple
- Increases CPI
- Use for rare case since stalling has performance effect

2. Pipeline hardware resource

- Useful for multi-cycle resources
- Good performance
- Sometimes complex e.g., RAM

3. Replicate resource

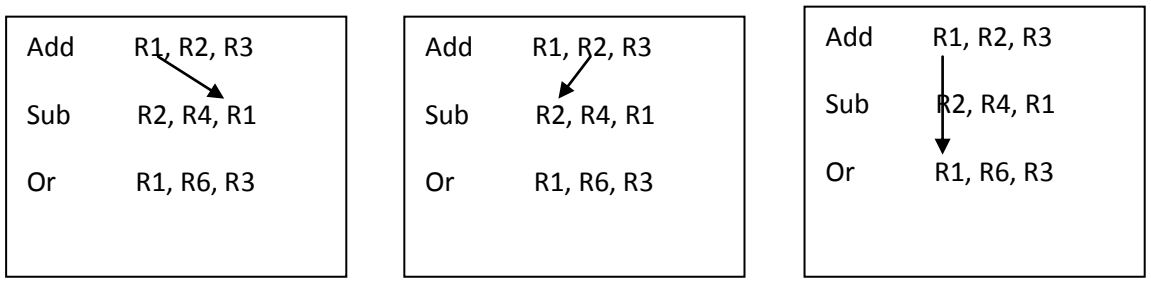
- Good performance
- Increases cost (+ maybe interconnect delay)
- Useful for cheap or divisible resources

Thus, Structural hazards are reduced with these rules:

- Each instruction uses a resource at most once
- Always use the resource in the same pipeline stage
- Use the resource for one cycle only

3.6.1.2 Data Hazards

These occur when at any time, there are instructions active that need to access the same data (memory or register) locations. It must appear as if they executed in sequential order. It can be of three types: 1. Read-After-Write (RAW) 2.Write-After-Read (WAR) 3. Write-After-Write (WAW). Figure 3.7 shows the three types of data hazards.



Read-after write (RAW)	write-after-read (WAR)	write-after-write (WAW)
True dependence (Real)	anti dependence (Artificial)	output dependence (Artificial)

Figure 3.7: Data Hazards [19]

- Dealing with Data hazards

  Simple Solution to RAW

  - Hardware detects RAW and stalls
  - Assumes register written then read each cycle
    - low cost to implement and simple
    - reduces IPC
  - Try to minimize stalls

    Minimizing RAW stalls

- Bypass/forward/short circuit
- Use data before it is in the register
  - It reduces/avoids stalls
  - It is complex
- Crucial for common RAW hazards

3.6.1.3 Control Hazards

A control hazard is when we need to find the destination of a branch, and can't fetch any new instructions until we know that destination. It occurs due to branch instructions. Figure 3.7 shows the Control hazards**.**

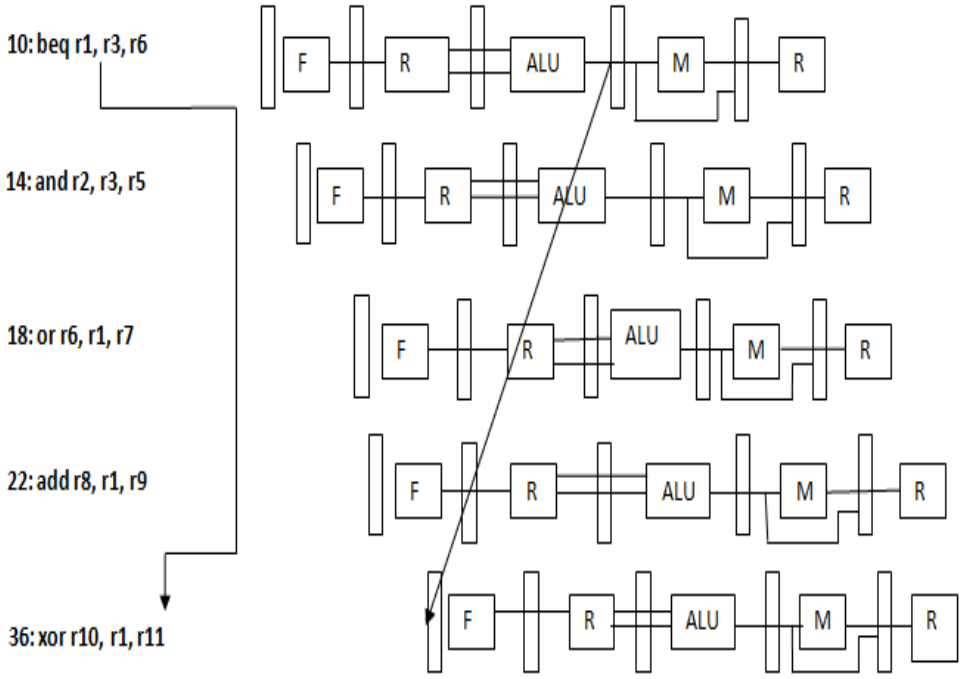


Figure 3.8: Control hazards [19]

1. Branch Instructions

- Branch instructions change the flow of program control.
- Branches follow one of two paths.
  - The fall through or not taken path.
  - The branch target stream or taken path.
- In general purpose code branches occur approximately used below 5 instructions.
- Branch instructions cause control hazards.
- Branch instructions reduce processor performance.

- In simple processors instructions from the sequential path are pre-fetched from an I-cache to ensure the pipeline is fully utilized.
- A taken branch incurs a misfetch penalty.
- Forecasting the outcome of a branch ahead of time is essential to improve Processor performance.

2. Branch Problems

   It consists of two sub-problems:

   - Generating the correct prediction.
   - In the case of a taken branch predicting the correct target.

3. Branch Prediction

   Superscalar processors attempt to achieve high performance by pre-fetching groups of instructions into an instruction buffer and issuing those instructions for execution as soon as the required operands are available. Even with out-of-order instruction issue, a high sustained instruction issue rate can only be achieved if the contents of the instruction buffer spans several basic blocks. This can only be achieved by dynamic branch prediction.

There are two types of Branch Prediction.

i. Static Branch Prediction: The direction of each branch is predicted before a Programs runs using either compile time heuristics or profiling.

ii. Dynamic Branch Prediction: The direction of each branch is predicted by Recording information, in hardware, of past branch history during a program's Execution and is therefore done at run-time

- Static Branch Prediction Static branch prediction involves making a prediction at compile time. Predictions are either based on:
  - Compile time heuristics or,
  - Profile information that has been obtained from previous runs of the program. Static Branch Prediction predicts in the ID stage of the pipeline.

a) A Heuristic Approach (Predict Not Taken).

- The simplest static branch prediction scheme ignores the presence of a branch and continues to fetch instructions from the sequential instruction stream.
- All branches are therefore predicted as not taken.
- Incorrectly fetched instructions are squashed when the branch is resolved.

- Using predict not taken, a correctly predicted branch incurs no penalty.

b) Static Branch Prediction -a Heuristic Approach (Predict Taken).

- A correctly predicted branch will always incur a penalty of at least one clock cycle while the branch target address is being computed.

c) Static Branch Prediction -a Heuristic Approach (Predict Not Taken / Predict Taken).

- Predicting not taken is slightly better than always predicting taken.
- Predict taken scheme is more complex to implement than the predict not taken scheme.

d) Static Branch Prediction -a Heuristic Approach (BTFNT)

- Backward Taken Forward Not Taken
- Is based on the premise that backward branches tend to close loops and are therefore probably taken.
- The strategy is simple to implement since it relies only on the sign bit of the branch displacement. Since the type and direction of a branch must be known before a prediction can be made, the prediction must be delayed until the ID stage of the Pipeline [20].

e) Static Branch Prediction -a Heuristic Approach (Encoding into op-code)

- Program heuristics based on compile time knowledge can be encoded into the opcode of the branch instruction.
- This 'branch-likely' or 'hint-bit' is used to provide the branch direction the next time that the branch instruction is encountered [21].

f) Static Branch Prediction -a Profiling Approach

- By examination of profile information that has been obtained from previous runs of the program.
- Branch behavior is bimodal.
- An individual branch is biased to one path [22].

▪ Dynamic Branch Prediction

It reduces branch penalties under hardware control. The prediction is made in the IF stage of the pipeline. The simplest dynamic prediction scheme is a branch prediction buffer or branch history table.

a) A Branch Prediction Buffer

- Is a small fast memory.
- Contains history information of the previous outcome of the branch as a Prediction field.
- 0 for not-taken.
- 1 for taken.
- Is indexed by the low order bits of the branch address either in IF or ID/IR.
- The branch target can then be accessed as soon as the branch target address is computed, but before the branch condition is available.

Disadvantages of Branch Prediction Buffer are:

a) Even if a branch is predicted correctly there will be a branch penalty of one cycle.

b) May access a prediction bit for another branch that has the same least significant address bits. To rectify this problem a tag could be added to hold the MSBs of the Branch instruction address. However, this addition would significantly increase the size of the buffer.

c) In many RISC pipelines the branch is resolved in the ID pipeline stage. In this Case the branch prediction information would be obtained during the same cycle as the branch is resolved is too late to be of any use. In MIPS the prediction might precede the branch resolution by one cycle.

d) One bit achieves Limited prediction accuracy. P&H suggest that a buffer with 500–100 entries and two bits of prediction information per entry is likely to predict the outcome of branches correctly about 86% of the time.

4. A Branch Target Cache (BTC)

BTC is a Branch Prediction Buffer with more information:

- Has an address tag of a branch instruction the high order bits.
- Stores the target address.
- Usually uses two-bit up-down saturating counters as a prediction field [23].
- Can be organized in different ways:
    - Direct mapped
    - Fully associative
    - Set associative

Following are the features of BTC:

- Need to be determined while each instruction is being fetched:

- If the instruction is a branch
- If the branch will be taken
- If the branch prediction is taken, what the target address is.
- If these requirements are met the processor can initiate the next instruction access as soon as the previous access is complete [24].

The Operation of BTC is as follows:

1 During the IF stage, the LSB of the PC are also used to access the BTC.
2 If the MSBs of the PC match the tag, the entry is valid.
3 If the branch is predicted as taken, the predicted branch target address is used to access the I-cache during the next cycle.
4 When the branch is finally resolved, the prediction is verified and the BTC entry updated.

The Prediction Mechanism is as follows:

The predict taken/not taken bits implement simple finite state machines that record the Past history of the branch.

1 Single bit prediction: The simplest implementation is a single bit recording what happened the last time the branch was executed. Alternatively, entries can be saved in the BTC by only recording branches that were taken last time.
2 Two-bit bit prediction: Most BTCs use a two-bit up-down saturating counter. The counter is incremented every time a branch is taken to a maximum of 112and decremented every time a branch is not taken to a minimum of 002 [25].
3 Three-bit bit prediction: Generally does not increase the accuracy of predictions. Thus BTC achieves prediction accuracy between 80 and 90%. For multiple instructions issue (MII) processors. Many instructions issued before a misprediction is detected. A mispredicted branch incurs a heavy penalty. A BTC therefore does not provide sufficient accuracy for MII processors. The other mechanism for sorting the branch problems are:

1 Aggressive Scheduling
   - HSA
   - IA-64
2 Branch Delay Slots
3 Multithreading

3.7 2-Stage Pipelining

In 2 stage pipeline as shown in figure 3.8 8 clock cycles are required for 2 instructions. If time required for each instruction is Tex, then execution time for 7 instructions with pipelining is (Tex/2)*8= 4*Tex.

Clock	1	2	3	4	5	6	7	8
Inst i	FI	EI						
Inst i+1		FI	EI					
Inst i+2			FI	EI				
Inst i+3				FI	EI			
Inst i+4					FI	EI		
Inst i+5						FI	EI	
Inst i+6							FI	EI

Figure 3.9: 2 stage pipeline with-respect-to instruction pipelines

3.8 3-Stage pipeline

In order to enhance the performance of the VLIW, three pipelining stages are used to divide the critical path thus increasing the maximum operating frequency of the VLIW. Three pipelining stages mean that there is latency in the output by three clocks

3.9 Types of Microprocessor Architecture

To achieve high performance for microprocessors, the concept of pipeline is introduced into microprocessor architecture. In pipelining, a microprocessor is divided into multiple pipe stages. Each pipe stage can execute an instruction simultaneously.

When a stage in the pipe has completed executing its instruction, it will pass the results to the next stage for further processing while it takes another instruction from its Preceding stage.

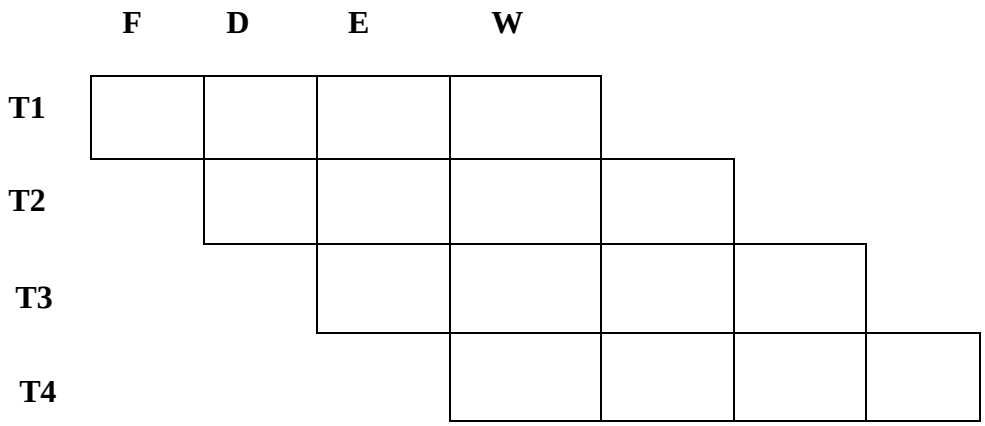


Figure 3.10: Diagram showing instruction execution for pipeline microprocessor [26]

Figure 3.10 shows the instruction execution for a pipeline microprocessor that has the four basic stages of pipe:

1. Fetch-This stage of the pipeline fetches instruction/data from instruction cache/memory.

2. Decode-This stage of the pipeline decodes the instruction fetched by the fetch stage. The decode stage also fetches register data from the register file.

3. Execute-This stage of the pipeline executes the instruction. This is the stage where the ALU (arithmetic logic unit) is located.

4. Writeback-This stage of the pipeline writes data into the register file. A pipeline microprocessor as shown in Figure 3.10 consists of basic four stages. These stages can be further subdivided into more stages to form a superpipeline microprocessor. A superpipeline microprocessor has the disadvantage of requiring more clock cycles to recover from a branch instruction compared to a fewer-stage pipeline microprocessor.

To achieve multiple instruction execution, multiple pipes can be put together to form a superscalar microprocessor. A superscalar microprocessor increases in complexity but allows multiple instructions to be executed in parallel. Figure 3.11 shows the instruction execution for a superscalar pipeline microprocessor. VLIW microprocessors use a long instruction word that is a combination of several operations combined into one single long instruction word. This allows a VLIW microprocessor to execute multiple operations in parallel. Figure 3.12 shows the instruction execution for a VLIW microprocessor. Although both superscalar pipeline and VLIW microprocessors can execute multiple instructions in parallel, each microprocessor is very different and has its own set of advantages and disadvantages.

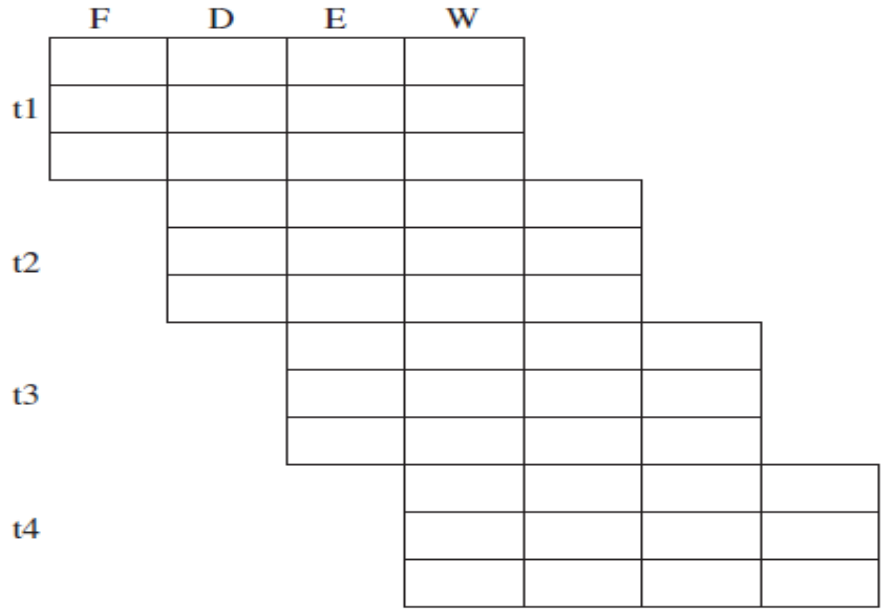


Figure 3.11: Diagram showing instruction executions for superscalar pipeline microprocessor

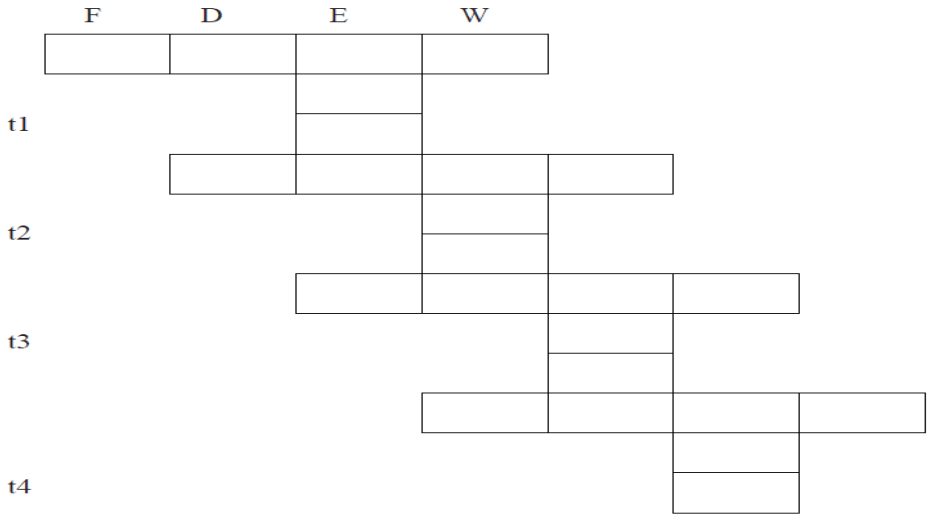


Figure 3.12: Diagram showing instruction execution for VLIW microprocessor. [27]

3.10 Instruction-Level parallelism

Instruction-level Parallelism (ILP) is a family of processor and compiler design techniques that speed up execution by causing individual machine operations, such as memory loads and stores, additions and floating point multiplications, to execute in parallel. The operations Involved are normal RISC-style operations, and the system is handed a single program written with a sequential processor in mind. Thus an important feature of these techniques is that like circuit speed improvements, but unlike traditional

multiprocessor parallelism and massive parallel processing, they are largely transparent to users. VLIWs and superscalar's are examples of processors that derive their benefit from instruction-level parallelism, and software pipelining and trace scheduling are example software techniques that expose the parallelism that these processors can use. Although small amounts of ILP has been present in the highest performance uniprocessors of the past 30 years, the 1980s saw it became a much more significant force in computer design. [28]. Several systems were built, and sold commercially, which pushed ILP far beyond where it had been before, both in terms of the amount of ILP offered and in the central role ILP played in the design of the system. By the early 1990s, advanced microprocessor design at all major CPU manufacturers incorporated ILP, and new techniques for ILP became a popular topic at academic conferences.

CHAPTER-4

FIELD PROGRAMMABLE GATE ARRAY

This chapter introduces about the FPGA concepts and FPGA Synthesis Flow. An FPGA is a device that consists of thousands or even millions of transistors connected to perform logic functions. They perform functions from simple addition and subtraction to complex digital filtering and error detection and correction.

4.1 Introduction to FPGA

A field programmable gate array (FPGA) is a semiconductor device that can be configured by the customer or the designer after manufacturing hence the name "field-programmable". Field Programmable gate arrays (FPGAs) are truly revolutionary devices that blend the benefits of both hardware and software. FPGAs are programmed using a logic circuit diagram or a source code in Hardware Description Language (HDL) to specify how the chip will work. They can be used to implement any logical function that an Application Specific Integrated Circuit (ASIC) could perform but the ability to update the functionality after shipping offers advantages for many applications. FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together" somewhat like a one chip programmable breadboard. Logic blocks can be configured to perform complex combinational functions or merely simple logic gates like AND and XOR. In most FPGAs, the logic block also includes memory elements, which may be simple flip flops or more complete blocks of memory. FPGAs blend the benefits of both hardware and software.

They implement circuits just like hardware performing huge power, area and performance benefits over softwares, yet can be reprogrammed cheaply and easily to implement a wide range of tasks. Just like computer hardware, FPGAs implement computations spatially, simultaneously computing millions of operations in resources distributed across a silicon chip. Such systems can be hundreds of times faster than microprocessor-based designs. However unlike in ASICs, these computations are programmed into a chip, not permanently frozen by the manufacturing process. This means that an FPGA based system can be programmed and reprogrammed many times.

FPGAs are being incorporated as central processing elements in many applications such as consumer electronics, automotive, image/video processing military/aerospace, base stations, networking/communications, super computing and wireless applications.

4.2 FPGA for Floating Point Computations

With gate counts approaching ten million gates, FPGA's are quickly becoming suitable for major floating point computations. However, to date, few comprehensive tools that allow for floating point unit trade offs have been developed. Most commercial and academic floating point libraries provide only a small number of floating point modules with fixed parameters of bit-width, area and speed [18]. Due to these limitations, user designs must be modified to accommodate the available units. The balance between FPGA floating point unit resources and performance is influenced by subtle context and design requirements. Generally, implementation requirements are characterized by throughput, latency and area.

1. FPGAs are often used in place of software to take advantage of inherent parallelism and specialization. For data intensive applications, data throughput is critical.

2. If floating point computation is in a dependent loop, computation latency could be an overall performance bottleneck.

4.3 FPGA Technology Trends

- General trend is bigger and faster.
- This is being achieved by increases in device density through even smaller fabrication process technology.
- New generations of FPGAs are geared towards implementing entire systems on a single device.
- Features such as RAM, dedicated arithmetic hardware, clock management and transceivers are available in addition to the main programmable logic.
- FPGAs are also available with the embedded processors (embedded in silicon or as cores within the programmable logic fabric).

4.4 FPGA Implementation

The FPGA that is used for the implementation of the circuit is the Xilinx Spartan 3E (Family), XC4VFX12 (Device). The working environment/tool for the design is the Xilinx ISE 13.1 is used for FPGA Design flow of Verilog code.

4.4.1 Overview of FPGA Design Flow

As the FPGA architecture evolves and its complexity increases. Today, most FPGA vendors provide a fairly complete set of design tools that allows automatic synthesis and compilation from design specifications in hardware specification languages, such as Verilog or VHDL, all the way down to a bit stream to program FPGA chips. A typical FPGA design flow includes the steps and components shown in Figure 4.1 Inputs to the design flow typically include the HDL specification of the design, design constraints, and specification of target FPGA devices. We further elaborate on these components of the design input in the following: Design constraints typically include the expected operating frequencies of different clocks, the delay bounds of the signal path delays from input pads to output pads (I/O delay), from the input pads to registers (setup time), and from registers to output pads (clock-to-output delay). In some cases, delays between some specific pairs of registers may be constrained. The second design input component is the choice of FPGA device. Each FPGA vendor typically provides a wide range of FPGA devices, with different performance, cost, and power tradeoffs. The selection of target device may be an iterative process.

The designer may start with a small (low capacity) device with a nominal speed-grade. But, if synthesis effort fails to map the design into the target device, the designer has to upgrade to a high-capacity device. Similarly, if the synthesis result fails to meet the operating frequency, he has to upgrade to a device with higher speed-grade. In both the cases, the cost of the FPGA device will increase in some cases by 50% or even by 100%. This clearly underscores the need to have better synthesis tools since their quality directly impacts the performance and cost of FPGA designs [29].

Design Entity

The basic architecture of the system is designed in this step which is coded in a Hardware description Language like Verilog or VHDL. A design module is split into two parts, each of which is called a design unit in Verilog. The module declaration represents the external interface to the design module. The module internals represents the internal description of the design module-its behavior, its structure, or a mixture of both.

Behavioral Simulation

After the design phase, create a test bench waveform containing input stimulus to verify the functionality of the verilog code module using a simulation software i.e. Modelsim SE for different inputs to generate outputs and if it verifies then proceed further,

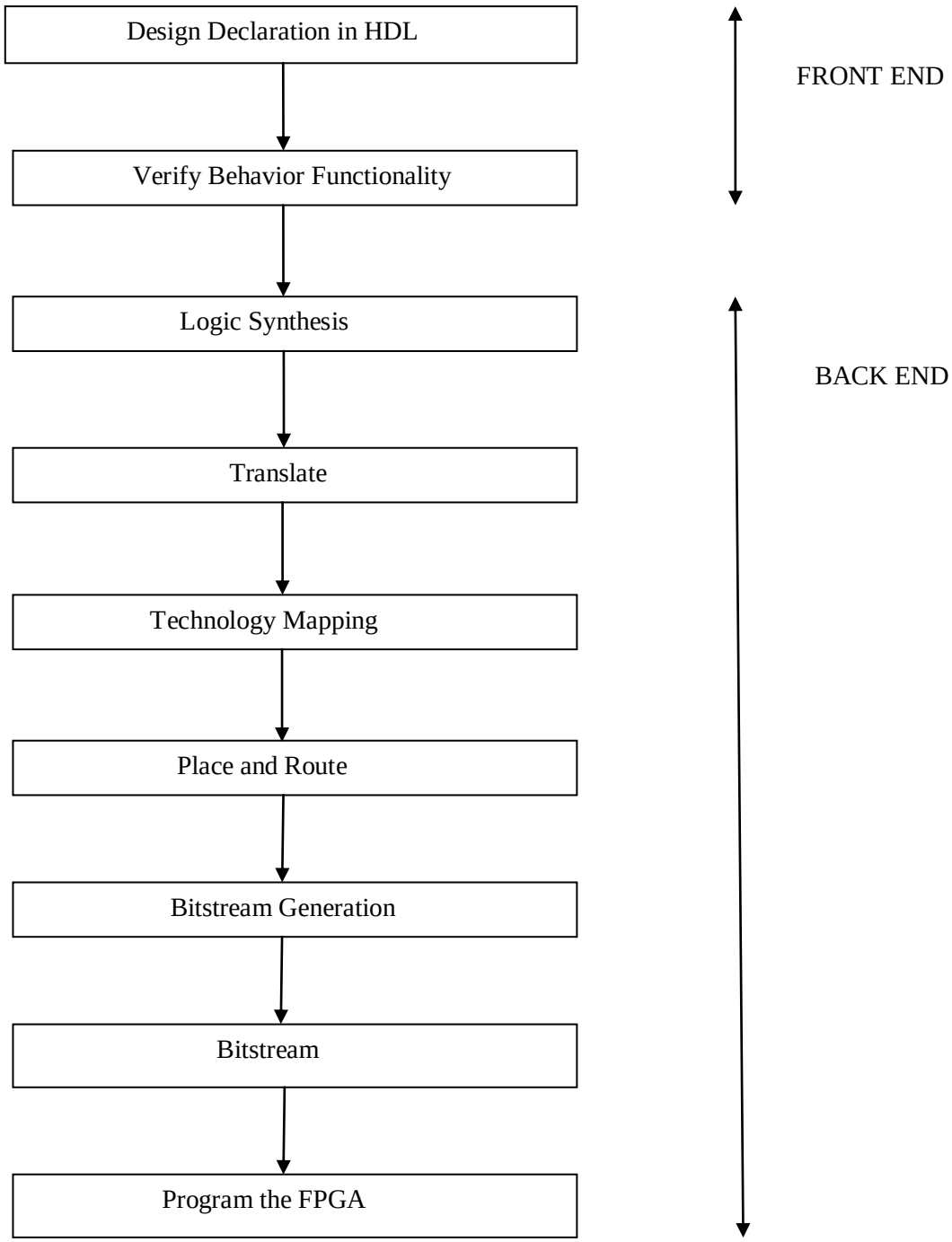


Figure 4.1: FPGA Design Flow [30]

otherwise modifications and necessary corrections will be done in the HDL code. This is called as the behavioral simulation

Design Synthesis

After the correct simulations results, the design is then synthesized. During synthesis, the Xilinx ISE tool does the following operations:

a) HDL Compilation: The tool compiles all the sub-modules in the main module if any and then checks the syntax of the code written for the design.

b) Design Hierarchy Analysis: Analysis the hierarchy of the design.

c) HDL Synthesis: The process which translates VHDL or Verilog code into a device netlist formats, i.e. a complete circuit with logical elements such as Multiplexer, Adder/subtractors, counters, registers, flip flops Latches, Comparators, XORs, tristate buffers, decoders, etc. for the design. If the design contains more than one sub designs, ex. to implement a processor, we need a CPU as one design element and RAM as another and so on, and then the synthesis process generates netlist for each design element. Synthesis process will check code syntax and analyze the hierarchy of the design which ensures that the design is optimized for the design architecture, the designer has selected. The resulting netlist is saved to an NGC (Native Generic Circuit) file (for Xilinx® Synthesis Technology (XST)).

d) Advanced HDL Synthesis: Low Level synthesis: The blocks synthesized in the HDL synthesis and the Advanced HDL synthesis are further defined in terms of the low level blocks such as buffers, lookup tables. It also optimizes the logic entities in the design by eliminating the redundant logic, if any. The tool then generates a 'netlist' file (NGC file) and then optimizes it. The final netIist output file has an extension of .ngc. This NGC file contains both the design data and the constraints. The optimization goal can be pre-specified to be the faster speed of operation or the minimum area of implementation before running this process. The level optimization effort can also be specified. The higher the effort, the more optimized is the design but higher effort can also be specified. The higher the effort, the more optimized is the design but higher effort requires larger CPU time (i.e. the design time) because multiple optimization algorithms are tried to get the best result for the target architecture.

Design Implementation

The design implementation process consists of the following sub processes:

1. Translation: The Translate process combines all the input netlists and constraints to a logic design file. This information is saved an NGD (Native Generic Database) file. This can be done using NGD Build program and .NGD file describes the logical design reduced to the Xilinx device primitive cells. Here, defining constraints is nothing but, assigning the ports in the design to the physical elements (ex. pins, switches, buttons etc) of the targeted device and specifying time requirements of the design. This information is stored in a file named UCF (User Constraints File). Tools used to create or modify the UCF are PACE, Constraint Editor Etc.

2. Mapping: The Map process is run after the Translate process is complete. Map process divides the whole circuit with logical elements into sub blocks such that they can be fit into the FPGA logic blocks. That means map process fits the logic defined by the NGD file into the targeted FPGA elements (Combinational Logic Blocks (CLB), Input Output Blocks (IOB)) and generates an NCD (Native Circuit Description) file which physically represents the design mapped to the components of FPGA. MAP program is used for this purpose.

3. Place and Route: Place and Route (PAR) program is used for this process. The place and route process places the sub blocks from the map process into logic blocks according to the constraints and connects the logic blocks. Example if a sub block is placed in a logic block which is very near to IO pin, then it may save the time but it may affect some other constraint. So tradeoffs between all the constraints is taken account by the place and route process .The PAR tool takes the mapped NCD file as input and produces a completely routed NCD file as output. Output NCD file consist the routing information.

4. Bitstream Generation: The collection of binary data used to program the reconfigurable logic device is most commonly referred to as a "bitstream," although this is somewhat misleading because the data are no more bit oriented than that of an instruction set processor and there is generally no "streaming." While in an instruction set processor the configuration data are in fact continuously streamed into the internal units, they are typically loaded into the reconfigurable logic device only once during an initial setup phase.

5. Functional Simulation: Post-Translate (functional) simulation can be performed prior to mapping of the design. This simulation process allows the user to verify that the design has been synthesized correctly and any differences due to the lower level of abstraction can be identified.

6. Static timing analysis: Three types of static timing analysis can be performed that are:

(i) Post-fit Static timing analysis: The timing results of the Post-fit process can be analyzed. The Analyze Post-Fit Static timing process opens the timing Analyzer window, which interactively select timing paths in design for tracing.

(ii) Post-Map Static Timing Analysis: Analyze the timing results of the Map process. Post Map timing reports can be very useful in evaluating timing performance. Although route delays are not accounted for the logic delays can provide valuable information about the design. If logic delays account for a significant portion (>50%) of the total

allowable delay of a path, the path may not be able to meet your timing requirements when routing delays are added. Routing delays typically account for 45% to 65% of the total path delays. By identifying problem paths, redesign the logic paths to use fewer levels of logic, tag the paths for specialized routing resources, move to a faster device, or allocate more time for the path. If logic-only-delays account for much less (35%) than the total allowable delay for a path or timing constraint, then the place-and-route software can use very low placement effort levels. In these cases, reducing effort levels allow to decrease runtimes while still meeting performance requirements.

(iii) Post Place and Route Static Timing Analysis: Analyze the timing results of the Post- Place and Route process. Post-PAR timing reports incorporate all delays to provide a comprehensive timing summary. If a placed and routed design has met all of timing constraints, then proceed by creating configuration data and downloading a device. On the other hand, identify problems and the timing reports; try fixing the problems by increasing the placer effort level, using re-entrant routing, or using multi-pass place and route. Redesign the logic paths to use fewer levels of logic, tag the paths for specialized routing resources, move to a faster device, or allocate more time for the paths.

(iv)Timing Simulation: Perform Post-Place and Route simulation after the design has been and routed. After the design has been through all of the Xilinx implementation tools, a timing simulation netlist can be created. This simulation process allows to see how the design will behave in the circuit. Before performing this simulation it will benefit to create a test bench or test fixture to apply stimulus to the design. After this an .NCD file will be created that is used for the generation of power results of the design.

CHAPTER-5

PROPOSED ARCHITECTURE (VLIW)

5.1 Proposed Architectures

The proposed architecture consists of 4-stages namely:

1. Instruction Fetch (IF),
2. Instruction Decode (ID),
3. Execution unit (EX) and
4. Write back (WB)

Figure 5.1 shows the block diagram of proposed architecture.

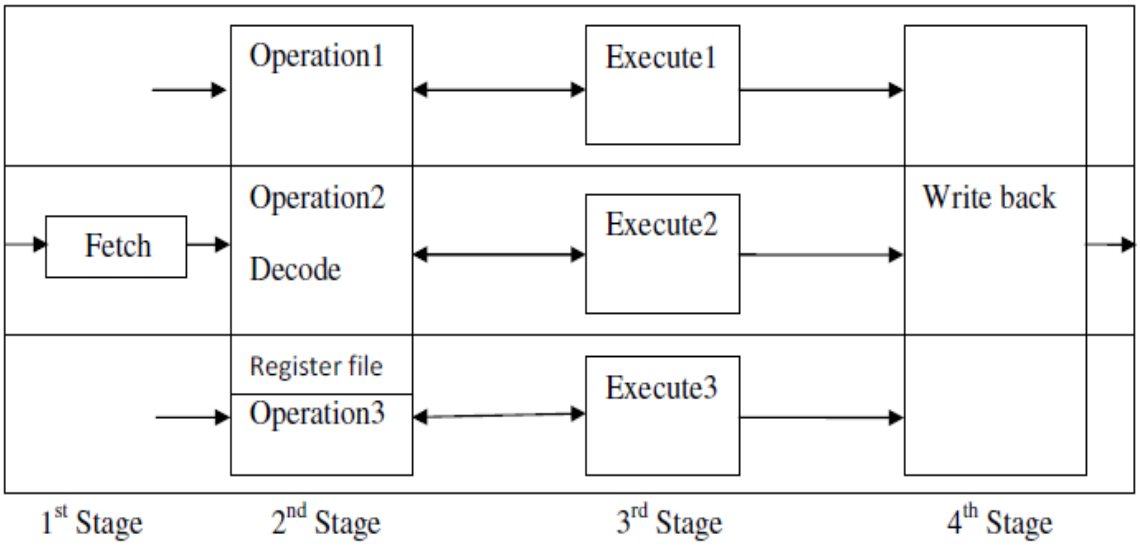


Figure 5.1: Block Diagram of Proposed Architecture. [31]

Detailed description of block diagram is as follows:

From the technical specification, the architectural specification is derived. This is a crucial step because the architecture of a design plays an important part in the performance capability and area utilization of the design. Figure 5.1 shows a VLIW architecture that can be used to represent the VLIW microprocessor. The microprocessor fetches instructions from an external instruction cache into its internal instruction buffers and decoders. The instruction is then passed on to multiple execution units which allows for multiple operations to be executed in parallel.

1. The VLIW microprocessor is architecture to take advantage of the pipeline technology.

2. Each 64-bit VLIW instruction word consists of three operations. To maximize the performance capability, the architecture is built to execute the three operations in parallel. Each operation is numbered and categorized as pipe1, pipe2, and pipe3 with pipe1 operating operation 1, pipe2 operating operation 2 and pipe3 operating operation 3.

3. Each operation is split into four stages: fetch stage, decode stage, execute stage, and write back stage. Four stages are chosen to keep the architecture simple yet efficient. The fetch stage fetches the VLIW instruction and data from external devices such as memory. The decode stage decodes the VLIW instruction to determine what operations each pipe needs to execute. The execute stage executes the operation decoded by the decode stage. The write-back stage (the last stage of the pipe) writes the results from the execution of the instruction into internal register.

4. All three operations share a set of sixteen 64-bit internal registers, which forms a register file. During the decode stage, data are read from the register file and during write-back stage, data are written into the register file. FPGA is used as it needs to have adequate elements and enough usable IO pins for implementation of the VLIW microprocessor core. The implemented microprocessor core is a 4 stage pipeline, 3 parallel pipes superscalar VLIW microprocessor [33]. The microprocessor core is designed with a shared register file with 16 registers accessible by all 3 pipes. Each register's width is the same as that of the data bus. Figure 5.1 shows the top level block diagram of the microprocessor core.

5.2 Technical Specification

In this step, the technical features and capability of the VLIW microprocessor are defined. The specifications will influence the architecture and micro-architecture of the Microprocessor. From the specifications, all design considerations are made with respect to meeting the specified technical requirements. A list of the technical specifications for the design and implementation of the microprocessor follows:

1. Microprocessor operates in 64 bits.

2. Area of design implementation must be kept to a minimum to reduce cost.

3. The microprocessor has sixteen internal registers, 64 bits each. This will form the Register file of the microprocessor. Each register is addressed using 5 bits, ranging from address R0000 for register 0 to address R1111 for register 15.The most Significant bit of the address is reserved for future expansion

4. Instruction sets include arithmetic operations, logical, and compare operations.

5.3 Micro-architecture Specification

Figure 5.2 shows the system level implementation utilizing the microprocessor core implemented on FPGA. The cache, prefetch, branch prediction is implemented external to the microprocessor core on the system level as the FPGA (implemented microprocessor core) has limited resource.The architecture and micro-architecture are closely related as both are the starting points on which a design is defined. In this step (micro-architecture specification), the block modules for the design are defined together with the top level intermodule signals.

1. Functional partitioning of the design
2. Intermodule connectivity signals
3. Intermodule signal naming

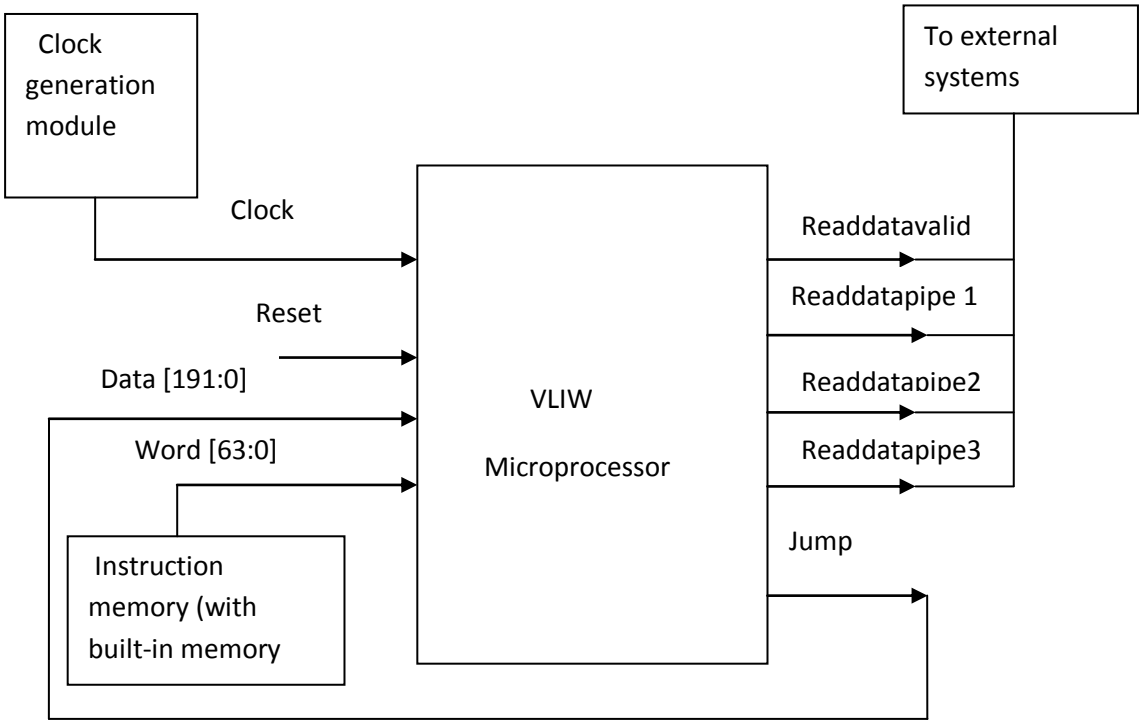


Figure 5.2: Diagram showing system level implementation utilizing microprocessor core on FPGA. [10]

5.3.1 Instruction Set of VLIW Microprocessor

When creating the instruction set for the VLIW microprocessor, the operations of arithmetic, load, read, and compare are considered and included in the instruction set. For the design of the VLIW microprocessor, an arithmetic and logic set of 16 operations is

created. The list of operations is shown in Table 5.1 with each operation represented by a 5-bit code, with the most significant bit being a reserved bit for future operation code expansion. A 64 bit custom instruction set VLIW. Microprocessor core is implemented on Spartan 3E (xc4vfx12) using the operations of arithmetic, logic, load, read and compare [34, 35,] creating a minimal customized instruction set of 16 instructions: each operation code is combined with the internal register address to form an

Table 5.1: Register Address for Internal Register of Register Files [10]

Register Name	Register Address
r0	R0000
r1	R0001
r2	R0010
r3	R0011
r4	R0100
r5	R0101
r6	R0110
r7	R0111
r8	R1000
r9	R1001
r10	R1010
r11	R1011
r12	R1100
r13	R1101
r14	R1110
r15	R1111

VLIW microprocessors commonly have anywhere up to 64 instructions or more. For ease of understanding, the VLIW microprocessor example is defined with only 16 instructions. [36]

Table 5.2: Operation Code for the VLIW Microprocessor Instruction Set [10]

Operation	Code
Nop	R0000
Add	R0001
Sub	R0010
Mul	R0011
Load	R0100
Move	R0101
Read	R0110
Compare	R0111
Xor	R1000
Nand	R1001
Nor	R1010
Not	R1011
Shift left	R1100
Shift right	R1101
Barrel shift left	R1110
Barrel shift right	R1111

Each operation code is combined with the internal register address to form an arithmetic or logic operation. Each operation consists of 5 bits for defining the operation code (as shown in Table 5.2) and 15 bits for the internal register addresses (as shown in Table 5.1). In total, an operation will consists of 20 bits. The instruction set used in this architecture consists of arithmetic instructions, logical instructions, memory instructions and branch

instructions. Table 5.3 shows the instruction set used in this architecture. Where xxxx denotes don't care.

Table 5.3: combination of operation code and internal register addresses to form an operation

Bits [19:15]	**Bits[14:10]**	**Bits[9:5]**	**Bit[4:0]**
Operation code	Source1 address	Sourec2 address	Destination address

1. Operation Code R0000- nop (no operation)

Bit format for operation code nop

Bits [19:15]	**Bits[14:10]**	**Bits[9:5]**	**Bits[4:0]**
Operation code	RXXXX	RXXXX	RXXXX

2. Bit Operation code R0001-add

Bit format for operation code add

Bits[19:15]	**Bits[14:10]**	**Bits[9:5]**	**Bits[4:0]**
Operation code	Src1	Src2	Dst

Destination= Src1+Src2

3. Operation code R0010-sub

Bit format for operation code nand

Bits [19:15]	**Bits[14:10]**	**Bits[9:5]**	**Bits[4:0]**
Operation code	Src1	Src2	Dst

Destination= Src1-Src2

4. Operation code R0011-mul

Bit format for operation code mul

Bits [19:15]	Bits[14:10]	Bits[9:5]	Bits[4:0]
Operation code	Src1	Src2	Dst

Destination =Src1*Src2

5. Operation code R0100-load

Bit format for operation code load

Bits [19:15]	Bits[14:10]	Bits[9:5]	Bits[4:0]
Operation code	Src1	Src2	Dst

6. Operation code R0101-move

Bit format for operation code move

Bits [19:15]	Bits[14:10]	Bits[9:5]	Bits[4:0]
Operation code	Src1	Src2	Dst

Output port = source1

7. Operation code R0110-read

Bit format for operation code

Bits [19:15]	Bits[14:10]	Bits[9:5]	Bits[4:0]
Operation code	Src1	Src2	Dst

8. **Operation code R0111-compare**

Bit format for operation code

Bits [19:15]	Bits[14:10]	Bits[9:5]	Bits[4:0]
Operation code	Src1	Src2	Dst

i. Source1 =Source2 → Branch to another instruction, a jump is required

ii. Source1 > Source2 → Bit 1 of destination register =1

iii. Source1<Source2 → Bit 2 of destination register =1

iv. Source1<= Source2 ⟶ Bit 3 of destination register =1

v. Source1>=Source2 ⟶ Bit 4 of destination register =1

vi. All other bits of destination register are set to 0.

9. Operation code R1000-xor

Bits [19:15]	**Bits[14:10]**	**Bits[9:5]**	**Bits[4:0]**
Operation code	Src1	Src2	Dst

This operation code is for xor function. The VLIW microprocessor perform an Xor function on data from internal registers specified by source1 and source2,and write the result into the internal register specified by destination.

10. Operation code R1001-nand

Bit format for operation code nand

Bits [19:15]	**Bits[14:10]**	**Bits[9:5]**	**Bits[4:0]**
Operation code	Src1	Src2	Dst

11. Operation code R1010 nor

Bit format for operation code nor

Bits [19:15]	**Bits[14:10]**	**Bits[9:5]**	**Bits[4:0]**
Operation code	Src1	Src2	Dst

12. Operation code R1011 not

Bit format for operation code not

Bits [19:15]	**Bits[14:10]**	**Bits[9:5]**	**Bits[4:0]**
Operation code	Src1	RXXXX	Dst

13. Operation code R1100 shift left

Bit format for operation code shift left

Bits [19:15]	Bits[14:10]	Bits[9:5]	Bits[4:0]
Operation code	Src1	Src2	Dst

14. Operation code R1101 shift right

Bit format for operation code shift right

Bits [19:15]	Bits[14:10]	Bits[9:5]	Bits[4:0]
Operation code	Src1	Src2	Dst

15. Operation code R1110 barrel shift left

Bit format for operation code barrel shift left

Bits [19:15]	Bits[14:10]	Bits[9:5]	Bits[4:0]
Operation code	Src1	Src2	Dst

16. Operation code R1111 barrel shift right

Bit format for operation code barrel shift right

Three operations have been used in operation 1, operation 2, and operation3. R indicates the reserved bits for future expansion; X indicates don't care for the corresponding operation.

Operation 1: add r0, r1, r2

Bit [19:15]	Bits [14:10]	Bits [9:5]	Bits[4:0]
R0001	R0000	R0001	R0010

Operation 2: sub r3, r4, r5

Bit [19:15]	Bits [14:10]	Bits [9:5]	Bits[4:0]
R0010	R0011	R0100	R0101

Operation 3 move r10, r8

Bit [19:15]	Bits [14:10]	Bits [9:5]	Bits[4:0]
R0101	R1010	RXXXX	R1000

VLIW instruction word:

Bit [64:60]	Bits [59:40]	Bits [39:5]	Bits[4:0]
RXXXX	R0001R0000R0001- R0010 Add r0,r1,r2 Operation 1	R0010R0011R0100- R0101 sub r3,r4,r5 operation 2	R0101R1010- RXXXXR1000 move r10,r8 operation 3

The three operations combined to form one VLIW instruction word allow the VLIW microprocessor to read one instruction but execute three operations in parallel. [37, 38]

CHAPTER-6

SIMULATION AND SYNTHESIS RESULTS

In This chapter introduces about the implementation and the simulation and synthesis results of 64-bit VLIW Microprocessor synthesized by Xilinx ISE 13.1 and Synopsys Design Compiler Tool.

6.1 Simulation

The use of Verilog for modeling is especially appealing since it provides formal description of the system and allows the use of specific description styles to cover the different abstraction levels (architectural, register transfer and logic level) employed in the design and design is verified through simulation.

6.2 Simulation Results of Instructions

In the processor, the instructions can be executed only through initializing the memory. I have written programs to exercise and test all the combinations of arithmetic, logical, and branch instructions. Arithmetic and logical instructions work with registers, meaning that the results of these instructions would store in destination register mentioned in the instruction. Whereas memory read and write operations are related to memory. So, the results from arithmetic and logical instructions can be correlated with the register values. The Figure 6.1, 6.2 and 6.3 show the waveforms for programs 1, 2 and 3 respectively and it shows all operations working perfectly. The Xilinx ISE 13.1 is used for implementation of this block diagram. The working Environment.

For the design is:

- Target Device: xc4vfx12-12sf363 (virtex-4)
- Tool Version: ISE 13.1
- Optimization Goal: Speed
- Design Strategy: Balanced
- Total Slices: 8723
- Total LUTs: 16979
- Modelsim: 10.1 student edition

6.3 Simulation Waveforms for Program

6.3.1 Simulation waveforms of verifying Barrel Shift Left, Subtract, Multiply and read

Figure 6.1 shows the simulation waveforms of Verifying Barrel Shift Left, Subtract, Multiply and read.

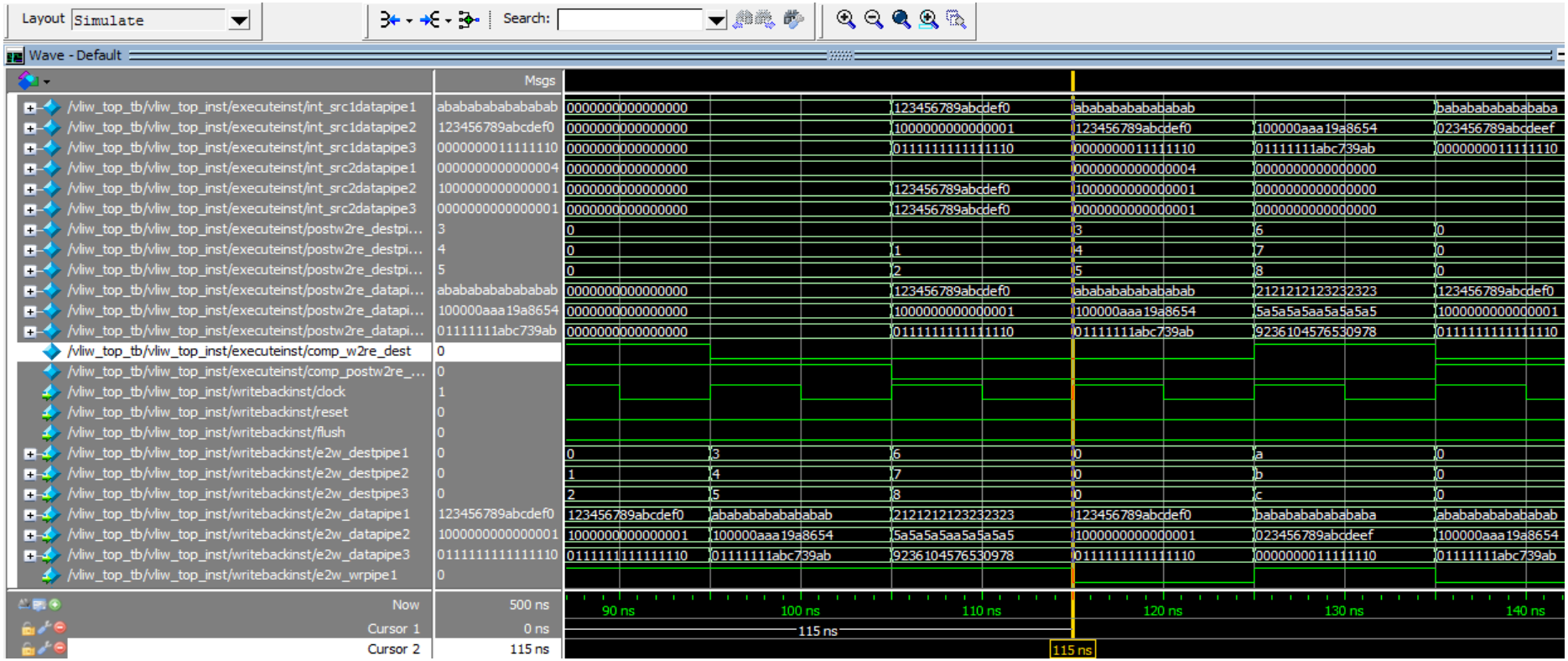


Figure 6.1: Simulation waveforms for Program 1

6.3.2 Simulation Waveforms for the Register Bypassing Conditions

Figure 6.2 shows the simulation waveforms of the register bypassing condition

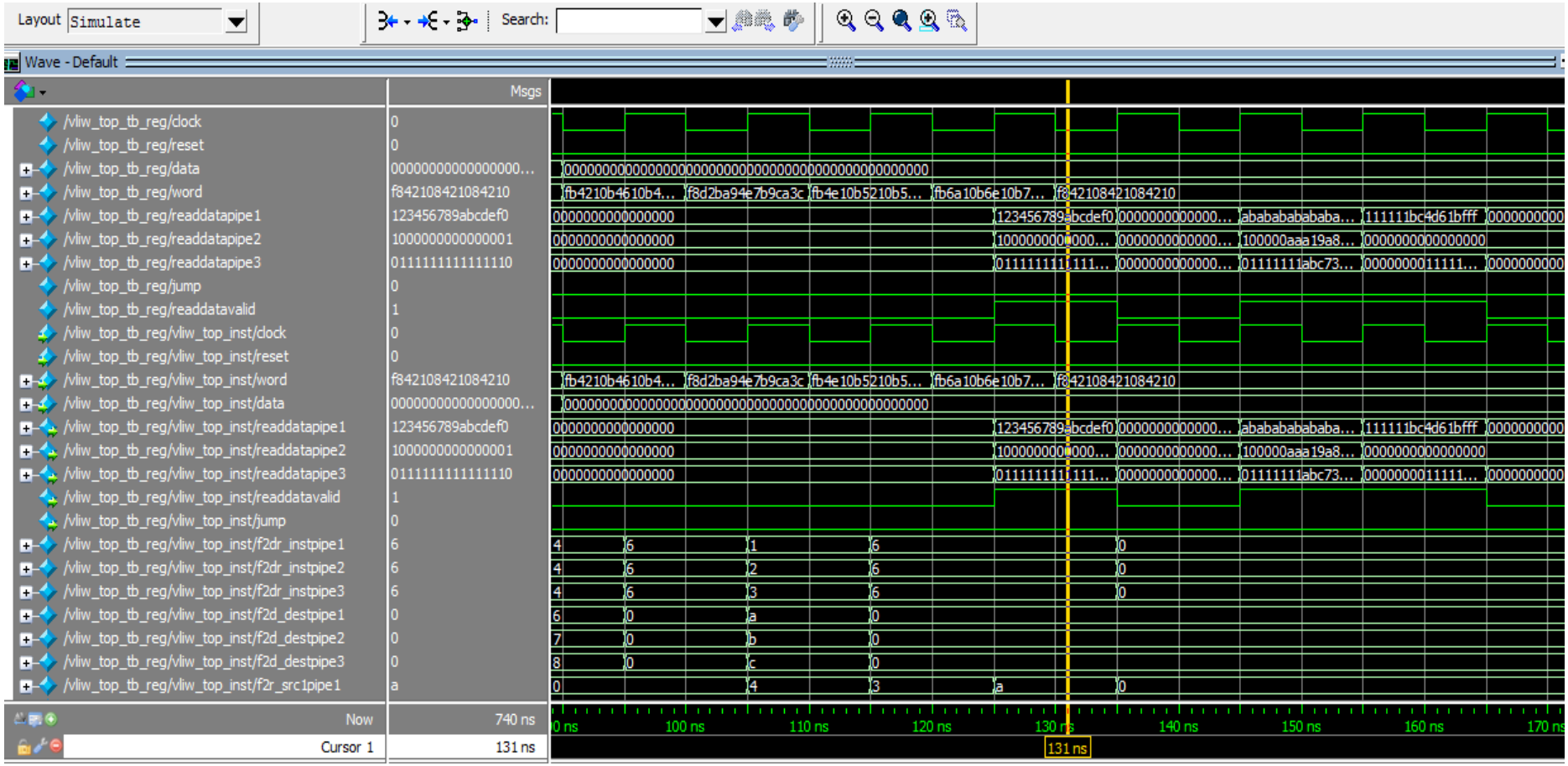


Figure 6.2: Simulation waveforms for Program 2

6.3.3 Simulation Waveforms for the Jump and Flush Condition

Figure 6.3 shows the simulation waveforms of jump and flush condition

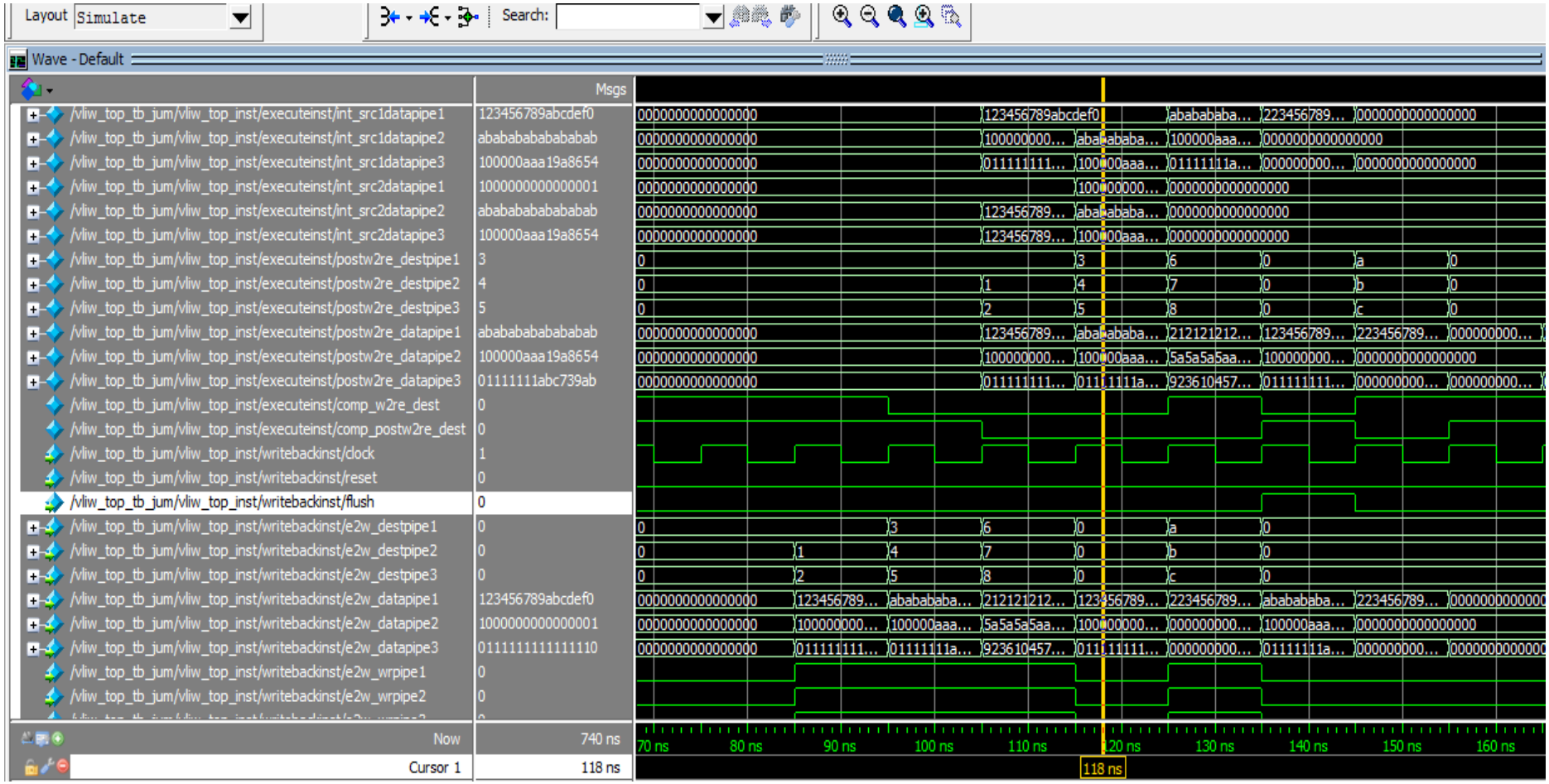


Figure 6.3: Simulation waveforms for Program 3

6.4 RTL TOP View

The RTL view of VLIW on FPGA is shown in Figure 6.4

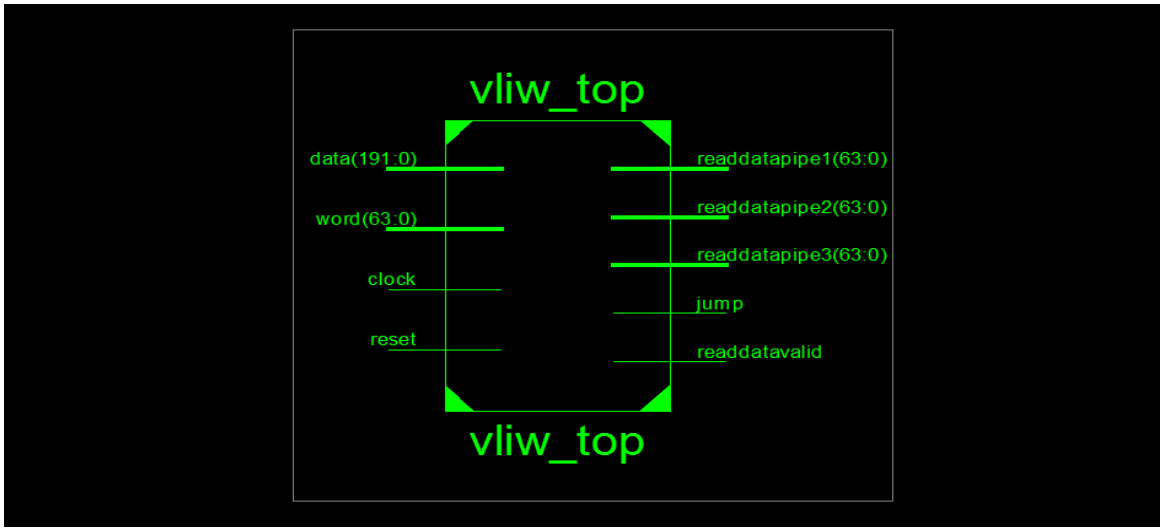


Figure 6.4: RTL VLIW TOP View

6.5 RTL Schematic

The RTL of proposed architecture is shown in Figure 6.5

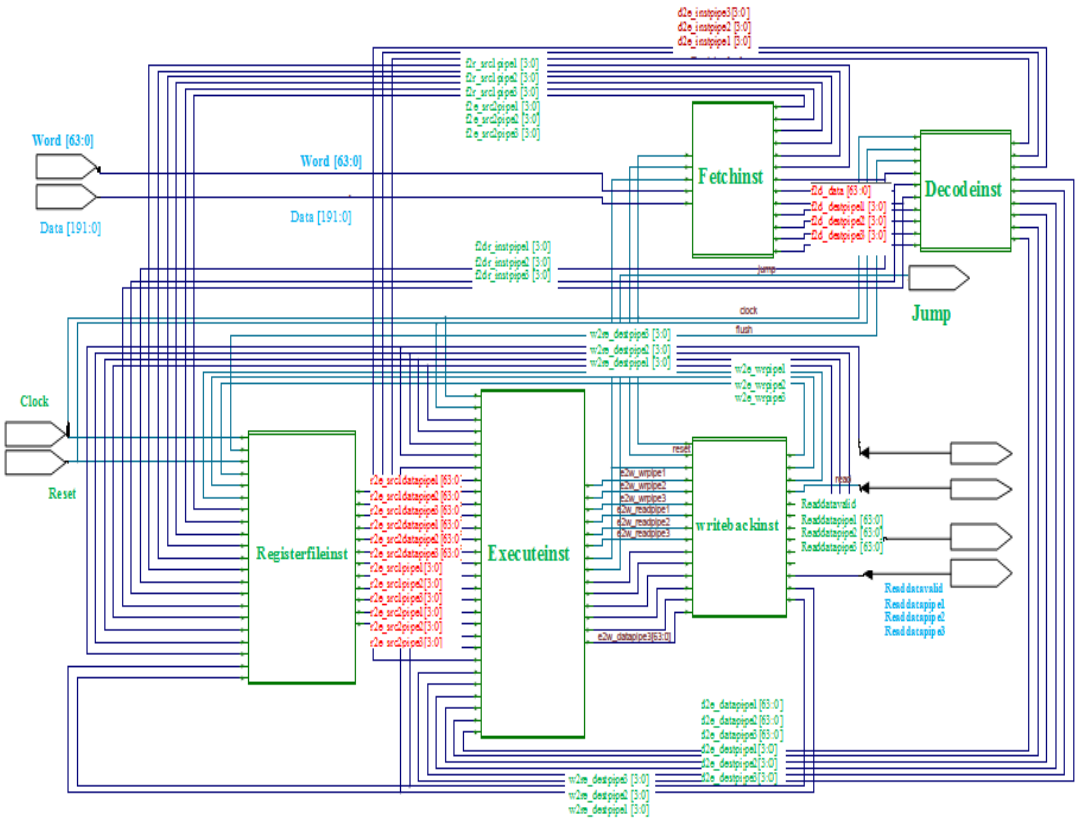


Figure.6.5 RTL: Schematic

6.6 Design Summary and Synthesis Result

The design summary of proposed architecture is shown in figure 6.6.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	2,728	10,944	24%	
Number of 4 input LUTs	16,934	10,944	154%	OVERMAPPED
Number of occupied Slices	8,490	5,472	155%	OVERMAPPED
Number of Slices containing only related logic	8,239	8,490	97%	
Number of Slices containing unrelated logic	251	8,490	2%	
Total Number of 4 input LUTs	16,979	10,944	155%	OVERMAPPED
Number used as logic	16,934			
Number used as a route-thru	45			
Number of bonded IOBs	436	240	181%	OVERMAPPED
Number of BUFG/BUFGCTRLs	1	32	3%	

Figure 6.6: Design Summary

As the pipeline stages are increased, maximum operating frequency is also increased.

6.6.1 Synthesis Results of 4-stage Pipelined 64-bit VLIW Microprocessor

a) Synthesis Results on Xilinx

Table 6.1 shows the synthesis report of 4-stage pipelined 64-bit VLIW Microprocessor on Xilinx.

Table 6.1 Shows the Synthesis Report of 64-Bit VLIW Microprocessor on Xilinx.

	Virtex-4 (xc4vfx12)
No. of Slices	8723/5472
No. of slice flip flops	2728/10944
No. of 4 input LUTs	16979/10944
Minimum Period	4.16ns
Maximum Frequency	240.38MHz

b) Synthesis Results on Design Compiler

Table 6.2 Shows the Synthesis Result for 4-stage pipelined 64-Bit VLIW Microprocessor on Design Compiler.

Table 6.2: Design Compiler Synthesis Result for 4-stage pipelined 64-Bit VLIW Microprocessor

Combinational Area	954056.312500
Noncombinational area	143742.281250
Total cell area	1097736.000000
Total Dynamic Power	19.9550 mW (100%)
Cell Leakage Power	335.0094 nW

CHAPTER-7

CONCLUSION AND FUTURE SCOPE

7.1 Conclusion

A design of 4-stage 64-bit VLIW microprocessor performing arithmetic, logical and compare operation and branch instructions is presented in this thesis. According to the basic principle of VLIW microprocessor, it is rationally divided into five main modules. Such as fetch module, decode module, register file, execute module, write back module. Each main module is reasonably divided again, and realized the function of every module based on the principle of FPGAs, so as to implement five main modules. At last, the whole function of VLIW microprocessor is completely finished. It completes the data of empty operation, addition, subtraction, multiplication, load, and move, read, comparison, XOR, NAND, NOR, NOT, shift left, shift right, barrel shift left, barrel shift right in VLIW microprocessor. Architecture is devised in order to felicitate the writing of codes in Verilog. The Verilog coding synthesis issues play a vital role in the speed-area optimality because RTL schematic depends heavily on how we have coded in Verilog. The design is simulated on Modelsim SE 10.1 tool and then synthesized by selecting device "xc4vfx12" and verified on FPGA Virtex-4. The proposed architecture is able to prevent pipelining from flushing when branch instruction occurs and able to provide halt support. It can be inferred from the synthesis report that proposed architecture offer speed which is approximately 240.38MHz. In this thesis have to added Synopsys results.

7.2 Future Scope

There are various ways in which this thesis work can be extended in near future like that:

1. Multiply instructions could be added.
2. Dynamic voltage scaling for further reduction in power can be included.
3. In the present design register file contains 16 registers in which each register stores 64 bit data. In future we can be designed for 32 registers in VLIW Processor.

REFERENCES

[1] Fisher, Joseph A, Paolo Faraboschi, and Cliff Young. “Embedded Computing a VLIW Approach to Architecture, Compilers and Tools,” New York: Morgan Kaufmann, 2004.

[2] Alex K.Jonaes, Raymond Hoare, Dara Kasic, Justin Stander, Gayatri Mehta, and josh Fazekas “A VLIW Processor With Hardware Functions Increasing Performance while Reducing Power,” IEEE Transactions on circuits and system-II: express briefs, Vol.53, no.11, Nov 2006.

[3] Rong-Jian Chen, Yi-Te Lai and Jui-Lin Lai, “Architecture Design and VLSI Hardware Implementation of Image Encryption/Decryption System Using Re-configurable 2-D Von Neumann Cellular Automata,” IEEE Proceedings of ISCAS, pp. 153-156, 2006.

[4] Hu Yue-li, Cao Jia-lin, Ran Feng and Liang Zhi-jian, “Design of High Performance Microcontroller,” IEEE Proceedings of HDP’04, pp. 25-28, 2004.

[5] Cem Savas¸ Bas Soy, Henning Manteuffel and Friedrich Mayer-Lindenberg, “SHARF: An FPGA-Based Customizable Processor Architecture,” IEEE International Conference on Field Programmable Logic and Applications, pp. 516-520, 2009.

[6] Yuichi Saito, Yukihiko Shimazu, Tom Shimizu, Kenji Shirai, Isao Fujioka, Yoshitetsu Nishiwaki, Junichi Hinata, Yoshiki Shimotsuma and Masayoshi Sakao, “A 1.71-Million Transistor CMOS CPU Chip with Testable Cache Architecture,” IEEE Journal of solid state circuits, Vol. 28, No. 1, Nov. 1993.

[7] J.P. Tual, M. Thill, C. Bernard, H.N. Nguyen, F. Mottini, M. Moreau, P. Vallet, “Auriga2: A 4.7 Million-Transistor CISC Microprocessor,” Design Automation Conference, and Proceedings of the ASP-DAC '95/CHDL '95/VLSI '95, 29 Aug-1 Sep 1995.

[8] Seiji Miura, Kazushige Ayukawa and Takao Watanabe, “A Dynamic-SDRAM-Mode-Control Scheme for Low-Power Systems with a 32-bit RISC CPU,” ACM ISLPED’OI, August 6-7, 2001, Huntington Beach, California, USA.

[9] Nick Richardson, Lun Bin Huang, Razak Hossain, Julian Lewis, Tommy Zounes and Naresh Soni, “The ICORE 520 MHz Synthesizable CPU core,” IEEE Proceedings of Design Automation Conference, pp. 640-645, 2003.

[10] Weng Fook Lee and Ali Yeon Md Shakaff "implementing a large Data Bus VLIW Microprocessor," Emerald Systems Design Center, Kompleks Sri Sg Nibong, Bayan Lepas, 11900, Penang, Malaysia School of Computer and Communication Engineering, University Malaysia Perlis, American journal of applied science, 2008.

[11] Michael R.Better, Johns.Fernando and Shaunp.Whalen "the history of the Microprocessor," Bell labs technical journal, Autumn 1995, pp. 44-52

[12] P. Pirsch, N. Demassieux, and W. Gehrke, "VLSI architectures for video compression -- A survey," Proceedings of the IEEE, vol. 83, pp. 220-245, 1995.

[13] V. L. Narasimhan, "A new course on supercomputers and parallel architectures," IEEE Transactions on Education, Vol. 38, pp. 340-345, 1995

[14] W. E. Mattis, "An advanced microprocessor course with a design component," SIGCSE Bulletin, Vol. 27, no. 4, pp. 60-64, 1995.

[15] S.-M. Moon and K. Ebcioglu. An efficient resource-constrained global scheduling technique for superscalar and VLIW processors. In the 25th Annual International Workshop on Microprogramming, Dec. 1992.

[16] Jean-Luc Gaudiot, Jung-Yup Kang, Won Woo Ro: "Techniques to Improve Performance Beyond Pipelining: Superpipelining, Superscalar, and VLIW," Advances in Computers 63: 2-35 (2005)

[17] Philips Semiconductors "Introduction to VLIW Computer Architecture".

[18] J. Hennessy and D. Patterson, Computer Architecture: A Quantitative Approach, Morgan Kaufmann, 4th edition, 2006.

[19] Hill Wood, Sohi, Smith and Vijaykumar and Moshovos. "Pipelining," in 1998.

[20] Smith, J. E. "A Study of branch prediction Strategies," Proceedings of the 8th Annual International Symposium on Computer Architecture, pp. 135-148, Minneapolis, Jun 1981.

[21] Ball, T. and Larus, J. "Branch Prediction for Free," Proceedings of the SigPlan93 Conference on Programming Language and Implementation, pp. 300-313, Jun 1993.

[22] Fisher, J. A. and Freudenberger, S. M. "Predicting Conditional Branch Directions from Previous Runs of a Program," Proceedings of the 5thInternational Conference on Architectural Support for Programming Languages and Operating Systems, Boston, Mass , pp. 85 –95, Oct 1, 2004.

[23] Lee, J. and Smith, J. "Branch Prediction Strategies and Branch Target Buffer Design," IEEE Journals on Computers, pp. 6 –22, Jan 1984.

[24] Perleberg, C. and Smith, J. "Branch Target Buffer Design and Optimization," IEEE Journal on Computers, vol. 4, pp. 396 –411, 1993. Intel Corp, 1999.

[25] Sechrest, S., Lee, C. and Mudge, T. "The Role of Adaptivity in Two-level Branch Prediction", Micro-28, Ann Arbor, Michigan, pp. 264 –269, Nov 1995.

[26] Arthur Abnous and Nader Bagherzadeh "Pipelining and Bypassing in a VLIW Processor," IEEE transactions on parallel and distributed systems, Vol. 5, no. 6, Jun 1994.

[27] Arthur Abnous and Nader Bagherzadeh "Architectural design and analysis of electrical and computer engineering," University of California, Irvine. Irvine CA 92717 PP.23

[28] B. Ramakrishna Rau, Joseph A. Fisher "Instruction-Level Parallel Processing: History, Overview and Perspective," To be published in The Journal of Supercomputing, Vol 7, No.1, Jan, 1993 © Copyright Hewlett-Packard Company 1992.

[29] D.Chen. J.Cong, and P. Pan, "FPGA Design Automation: A Survey," Foundations and Trends in Electronic Design Automation, 2006.

[30] V. P. Nelson, "VLSI/FPGA Design and Test CAD Tool Flow in Mentor Graphics," Feb 15, 2006

[31] Weng Fook Lee, Azrul Halim, Nor Hisham, Yap Vooi Voon, Lo Hai Hiung, Patrick Sebastian. "Implementation Results on Register Bypass Condiions of an n-Parallel Pipes Superscalar Pipeline Microprocessor Core on FPGA,"

[32] McFarling, S. and Hennessy, J. "Reducing the Cost of Branches", ACM 13th International Symposium of Computer Architecture, pp. 396 –403, Jun 1986.

[33] John L Hennessy and David A. Patterson. Computer Organization and Design: The Hardware/Software Interface. Morgan Kaufmann Publication.2003

[34] Weng Fook Lee, 2000. VHDL Coding and Logic Synthesis With Synopsys. Academic Press Publication.

[35] IBM Corp. 2000. Power PCTM Microprocessor Family: The Programming Environments for 32-Bit Microprocessors.

[36] Advanced Micro Devices Corp, 1997. Am186 and Am188 Family Instruction Set Manual.

[37] Half hill, Tom R. “VLIW Microprocessors,” Computer World, Feb. 14, 2000.

[38] Lee, Wang Fook. Verilog Coding for Logic Synthesis. New York: John Wiley, 2003.