# Simple Vector Processor Modeled with VHDL

Osvaldo Espinosa Sosa [1] , Luis Villa Vargas[2] and Oscar Camacho Nieto [1]

[1] Centro de Investigación en Computación-IPN, Av. Juan de Dios Batíz, esquina con Miguel Otón de Mendizábal, México, D.F., 07738. México
espinosa@cic.ipn.mx, oscarc@cic.ipn.mx
[2] Instiuto Mexicano del Petróleo, Eje Central Lázaro Cárdenas No. 152 Col. San Bartolo Atepehuacan, Delegación Gustavo A Madero, México, D.F.
lvilla@imp.mx

**Abstract.** Vector architectures have proven to be the best choice if we want to achieve high performance when executing numerical applications and multimedia. In this paper we describe a project whose main goal is to obtain a detailed description of a traditional vector processor using VHDL Hardware Description Language. This work will be very useful for direct application on computer architecture and digital systems courses at universities. Research on power consumption and improved architectures can be benefited also.

**Keywords:** Vector Processor, Hardware Description Language, VHDL.

## 1 Introduction

Vector processors are architectures that exploit parallelism at data level. They can operate on numeric arrays of data called vectors. One instruction can add or multiply two vectors containing the same number of data in order to obtain one vector containing results [1]. Vector instructions have several interesting properties such as:

- Each vector instruction represents several scalar instructions, this permit to reduce fetch and decode processes and the bandwidth required by these steps.
- Vector operations imply independence among results computation, this way no data dependence need to be checked. Dependences arise between two vector instructions not among vector elements.
- Instructions that access memory locations have regular patterns that permit design memory schemes with high grade of interleaving in order to obtain high bandwidth with low latencies.
- Because vector instructions operate in a predetermined way, control hazards are avoided.

It is clear that vector code can run faster than scalar code. This characteristic allows vector processors to execute numerical applications in a more efficient way that scalar computers. Computer architecture courses include vector processing as a topic. Study

of these architectures are important at graduate and undergraduate levels. Hardware Description Languages such as VHDL allow to obtain a precise description of digital circuits facilitating comprehension process. Modern software tools include capture, synthesis and simulation facilities. Students can be benefited using the mentioned software and its advantages.

## 2 Instruction Set Architecture

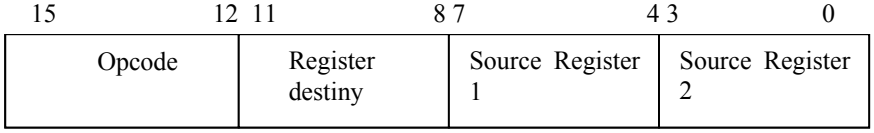
Design process begins with definitions about the instructions that processor can execute. Instructions can be divided into scalar and vector:

| Instructions | scalar | Vector |
|---|---|---|
| Arithmetic | add,sub | ADDV, SUBV, MULV |
| Memory | ld, st | LV, SV |
| Branch | bc,bz | |
| Control | clrc, setc | VEQ |

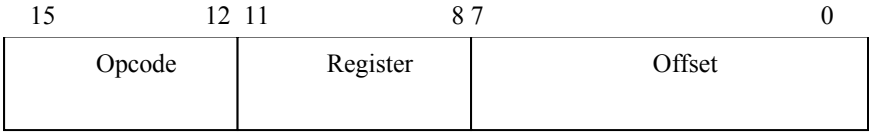
According to Load-Store architectures the syntax proposed is presented below:

| | |
|---|---|
| **add *rd, rf1,rf2*** | Adds registers rf1 and rf2. Result will be stored in rd. |
| **sub *rd, rf1,rf2*** | Substracts rf2 from rf1. Result will be stored in rd. |
| **ld *rd,dir*** | Load rd with content of memory location dir. |
| **st *rd,dir*** | Stores in memory location dir the content of *rd*. |
| **bc offset** | If carry flag is set, PC=PC+offset |
| **bz offset** | If zero flag is set PC=PC+offset |
| **clrc** | Clears carry flag |
| **setc** | Sets carry flag. |
| **add v1,v2,v3** | Adds vectors v2 and v3, result will be stored in v1. |
| **subv v1,v2,v3** | Substracts v3 from v2, result will be stored in v1 |
| **mulv v1,v2,v3** | Multiply v2 and v3, result will be stored in v1. |
| **lv v1,r1** | Load vector pointed by r1 in v1. |
| **sv v1,r1** | Store vector v1 at location pointed by r1. |
| **veq v1,v2** | Compares v1and v2, result will be stored in a mask. |

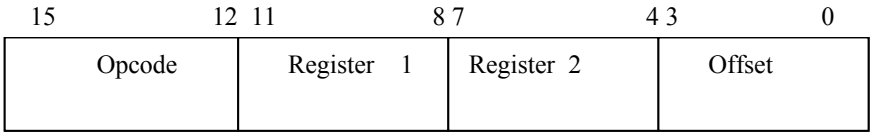
Arithmetic instructions have the following format:



Memory operations are coded as follows:




Branch Instructions and compare correspond to :




Other instructions such as control can only use de opcode field of format instruction.

## 3 Processor Architecture

Basically the vector processor has two sections [2]. It has one scalar processor and one vector processor as showed in figure 1.

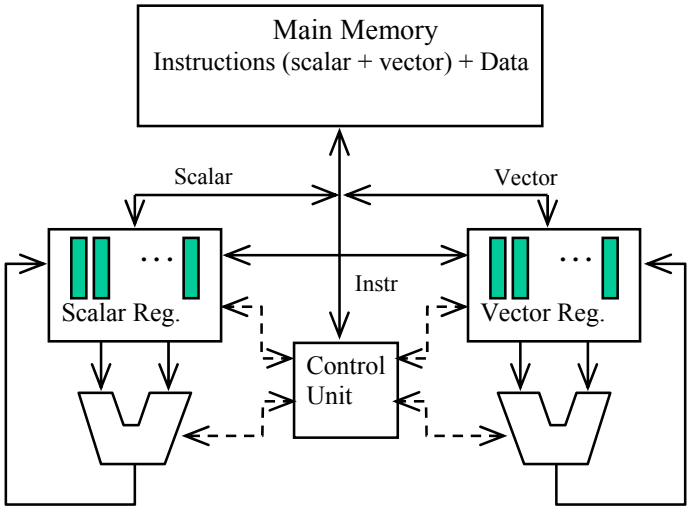



**Fig. 1.** Basic architecture of vector processor

### 3.1 Memory system

The first component is the memory system, and is normally the most expensive component in real vector computers because desired characteristics. It is desirable that memory system delivers one data each clock cycle in order to process vectors and generate results as soon as possible, so we need high bandwidth if we want to saturate the occupancy of functional units [3]. An interleaved memory scheme is used to obtain maximum bandwidth reducing the latency observed by memory accesses. Interleaved memory is shown in figure 2.

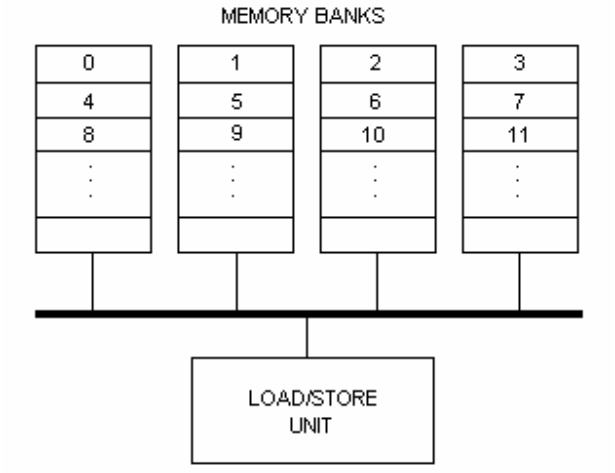



**Fig. 2.** Interleaved memory system

### 3.2 Register file

The register file in our design of vector processor is a component that includes eight vector registers of eight elements each. We are using short registers because we only want to show the mechanics of using this type of structures, real machine implementations use 64 or 128 elements per vector register. Resource restrictions on programmable devices affect also at implementation time. Vector registers interact with load/store unit to transfer data from memory to registers and from registers to memory. It is important to consider the number of read and write ports in order to diminish possible conflicts called structural hazards. At least we must include one write port and two read ports. Figure 3 shows the register file.

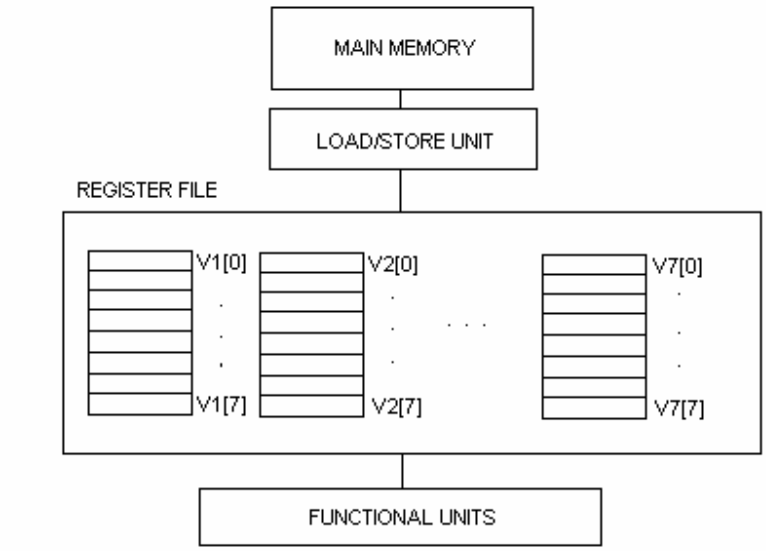



**Fig. 3**. Vector register file

### 3.3 Functional units

Figure 4 shows how functional units and register file are connected.

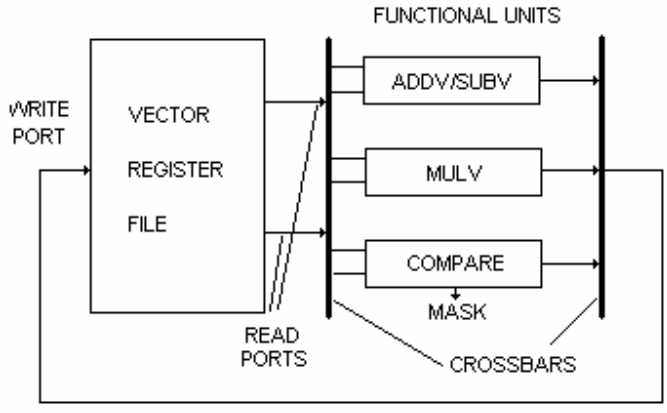



**Fig. 4**. Register file and functional units

Functional units in vector processors must be deeply pipelined if we want to process so much data as memory can deliver. Functional units must generate results at the same rate of memory transactions, of course one result is produced every clock cycle. We are adding several functional units of integer type as shown in figure 4.

## 4 Methodology

This project has been captured using software tools such as ISE WebPack 7.0 from Xilinx company. The first step was to capture VHDL modules corresponding to each processor section, after that, it is necessary to perform synthesis of modules. Once we have finished, we can enter to simulation process and then validate the correctness of circuitry. If all steps run successfully we could implement the design in a programmable logic device such as FPGA (Field Programmable Gate Array) [4].

## 5 Conclusions and future work

In this paper we describe a simple architecture for vector processing. This project has been captured and simulated, showing correct functionality. Hardware Description Languages allow us to design circuits in a fast way, reducing development times. The advantages of VHDL language permit to students to understand and modify the architecture in order to achieve the goals proposed in computer architecture courses. Future work will include description of advanced architectural techniques such as out of order execution, decoupling etc.

**Acknowledgments**. We would like to thank to Instituto Politécnico Nacional (IPN) for its economical support for the development of this research.

## References

[1] J.L. Hennessy and D.A. Patterson. Computer Architecture: a quantitative approach. Apendix G. Morgan Kauffman . 2003.
[2] F.Pardo Carpio. Arquitecturas avanzadas. Apuntes de la Universidad de Valencia, España. 2002.
[3] J.L. Hennesy and D.A. Patterson. Computer Organization and Design. Ed. Morgan Kauffman
[4] F. Pardo, J. A. Boluda. VHDL Lenguaje para síntesis y modelado de circuitos Ed. Alfaomega