

SPEED ENHANCEMENT OF THE VLIW ARCHITECTURE WITH AN INVOLVED LOAD INSTRUCTION

Alok Kumar Pani and Ratnam V. Raja Kumar

Department of Electronics and Electrical Communication Engineering, I.I.T. Kharagpur
Kharagpur, West Bengal, India. PIN-721302
E-mail: {alok_pani, rkumar}@ece.iitkgp.ernet.in

ABSTRACT

The VLIW architecture based processors have been a popular choice for digital signal processor implementations for its obvious advantages of simplicity and speed. Today's general purpose DSPs keep data memory and instruction memory separate to exploit the advantages of Harvard architecture in addition to gaining from the instruction level parallelism available in programs through the VLIW approach. In long duration processes like modems we make many multiplications on the data that is fetched from data memory. In this paper we propose a class of instructions having Register Indirect type addressing mode which will fetch data from the main memory two at a time, do some operations like multiplication and then load the result in a register. Since in a small length instruction, the operation involves main memory we call it as an involved load. To show the effectiveness of this instruction a 64QAM demodulator is taken as an example, which involves a pulse shaping (FIR) filter in it. The speed improvement achieved thereby is 15% without considering any static scheduling techniques.

1. INTRODUCTION

VLIW processors are used in communication related DSP applications because of their simplicity of hardware and exploitability of instruction level parallelism. The present state of the art is very much matured. Gaining of the speed has become essential because of the challenge thrown by the miniaturized implementation of the second and future generations of communication systems. New generations of digital signal processors based on VLIW architecture are now competitive compared to ASICs in terms of performance and price. The problem at hand is "Optimization of VLIW architecture for modem applications". Hence speed enhancement has been

emphasized in this paper. An instruction, namely Fetch, Multiply and Load (FML) has been introduced which is equivalent to two load instructions and one multiply instructions put together. It has been shown that this conglomeration saves 15% of clock cycles in a demodulation example.

This paper is organized as follows. Section 2 presents the VLIW architecture in connection to DSP applications, while Section 3 introduces the proposed involved load instructions and the architectural modifications needed in TMS320C62X to execute the instructions. Section 4 presents a communication receiver as an example for studying the effectiveness of the modification while some concluding remarks are made in Section 5.

2. THE VLIW ARCHITECTURE

The VLIW architecture is an extension of the RISC philosophy. In RISC we pipeline operations in a way to get one instruction executed per clock cycle. In VLIW we issue more than one instruction at a time and get more than one instruction per clock cycle as throughput. Instruction scheduling is done statically, that is at compile time, as a result of which we get tremendous simplicity of hardware. VLIW uses multiple independent processing elements to carry out instructions in parallel. Parallelism in code is uncovered by unrolling the loops and scheduling codes in the unrolled basic blocks. When loop unrolling is not sufficient the branches other than loops are handled by a method named 'trace scheduling'.

Architecture of the Texas Instruments' TMS320C62X DSP is used as an example to demonstrate the effectiveness of the proposed method. In this processor a maximum of 8 instructions of 32 bits each can be issued at a time. It has been pipelined in such a fashion that the throughput is maximum at 8 instructions per clock

cycle. It has separate program and data memory featuring HARVARD architecture along with VLIW. The number of instructions issued per clock cycle cannot be made very high (say, 50) in order to increase the speed because of the following limitations [2].

- Limited instruction level parallelism in programs
- Difficulties in building the underlying hardware
- Limitations specific to VLIW implementation

These limitations can be explained as follows. Firstly, for example, if the pipeline depth is 6 and 8 instructions are issued at a time; it results in to 48 independent instructions at a time to fully utilize the hardware. This amount of instruction level parallelism is hard to be found in processes belonging to communication systems. Secondly, the hardware complexity comes from the high memory bandwidth and high register bandwidth demand. In order to improve the speed we will not be able to afford the commonly used resources like multiple port register file and multiple port memory, because it consumes a lot of resources like area and power. Similar is the case with cache memory, where we cannot let a large real estate to be used.

Thirdly, there are technical and logistical problems in the VLIW model. Static scheduling makes the code size large and the unused functional units in a clock cycle cause wastage of some bits during instruction encoding (instruction encoding means putting together instructions for execution). The major logistical problem the VLIW processor faces is binary code incompatibility [2]. The number of instructions issued at a time and the functional unit latencies are different for different versions of the processor. So the instructions for one processor and the instructions for the next version of the processor are very different. As a result, the code for one application is so different from the code for the same application for the next version of the processor that the code for the later may be called as a new version of code. As an answer to the code incompatibility problem, a translator program may be used, which translates the object code of one machine to the object code of the next version of the machine. This is called object code translation. It can be concluded that a processor where multiple instructions are issued at a time, like VLIW has the potential to extract some amount of parallelism from less regularly structured code and it has the ability to use a less expensive memory system. Compiler can support to exploit Instruction Level Parallelism by detecting and eliminating data and control

dependencies among instructions. This level of fine grained parallelism is hard to achieve in a multiprocessor approach. The architectural diagram of the VLIW is as shown in Fig.1. It works as follows.

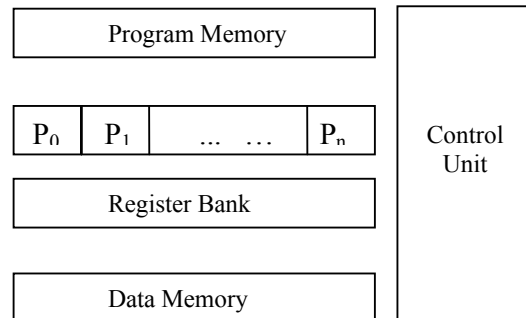


Fig.1 Architectural Diagram of a VLIW Machine

The control unit in the VLIW machine fetches multiple instructions from the program memory in the form of a very long instruction word. With the help of multiple independent processing elements in the ALU and the multiple port register banks, processing is done on the data fetched from the Data Memory. The whole process is pipelined to do instruction Fetch, Data load, Instruction Decode and Execute, result load and data store in a lock-step fashion, to exploit the parallelism in any process.

3. THE PROPOSED CLASS OF INVOLVED LOAD INSTRUCTIONS

Finite Impulse Response (FIR) filters are implemented in many communication and DSP related systems. These include load from main memory or cache and multiplication with filter coefficients from a lookup table. Our study of the TMS320C62X DSP gives us the fact that separate multiply and load instructions are provided. The result of a multiplication operation is generally added to a register rather than to be stored in a memory location. We propose an instruction namely Fetch, Multiply and Load (FML), which will fetch data from two memory locations by use of register indirect addressing, then multiply those two and the result is loaded into a register for further operation on that. We categorize this class of instructions as involved load instruction.

The Involved load instruction needs five registers for its operation. Two of them are base registers, two are offset registers and one is needed for the result of the operation to be loaded with. The instruction will always deal with bytes in the main memory. We have

to see that the instruction length does not exceed some specified minimum. Some Architectural modifications are needed for the implementation of FML instruction. A typical modification is to double the number of Data Address paths. Already two address calculation units (Data address paths) exist in the TI's processor TMS320C62X, which has been taken as an example. Two more address calculation units are to be placed. The multiplication units are to be modified to be able to execute the FML instruction along with the 'multiply' operation. The typical FML instruction format is as shown in Fig.2. This is a three-address instruction format. Each base-offset register pair specifies a location in main memory. The 'Dest' specifies the register where the result is loaded in. The reader is requested to refer [3] for other entries in the instruction format. The modified architectural diagram is as shown in Fig.4. The necessary modifications are shown shaded.

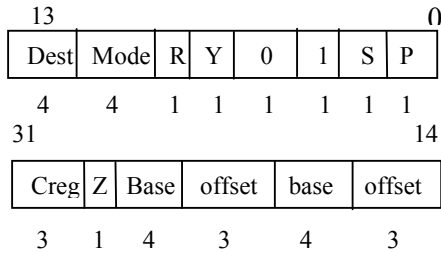


Fig. 2 The FML Instruction Format

4. 64-QAM DEMODULATION: AN EXAMPLE

To show the effectiveness of the processor with the inclusion of the FML instruction in reducing the number of clock cycles, a typical example of 64-QAM demodulation is considered. The block schematic is as shown in Fig.3. It has a 64 point FIR filter built into it for pulse shaping. 4 time samples per cycle of the carrier is considered. The carrier is assumed to be synchronized and then the number of samples per symbol = 16. The demodulation program has also been written in the TMS320C62X assembly language. In the implementation of the demodulator pulse shaping is the crucial portion from the computational complexity considerations. This part of the code forms the innermost loop. To process one sample of the signal we make 64 multiplications and 63 additions as per the following equation [4].

$$w(t) = \sum_{k=0}^{63} p(k)d(t-k) \quad (1)$$

The 64 FIR filter coefficients are stored in fixed locations in a circular queue facility in the architecture. 64 data samples are also get stored in a circular queue. Each time an output sample is generated the data queue is modified. One sample is over written to replace the oldest data of the queue. These two queues are multiplied location by location and accumulated to obtain the output sample. The innermost loop consists of four load instructions, two multiplications, one add and additional instructions to saturate the result of addition. It has been found that approximately 27000 clock cycles are needed to carry out one of the QAM symbol decisions. The same program when modified including the FML instruction a speed increase of more than 15% has been achieved. This has happened because in the innermost loop it could be managed with 7 groups of instructions, involving 22 clock cycles instead of 10, involving 26 clock cycles at no expense in latency. No loop unrolling has been exploited.

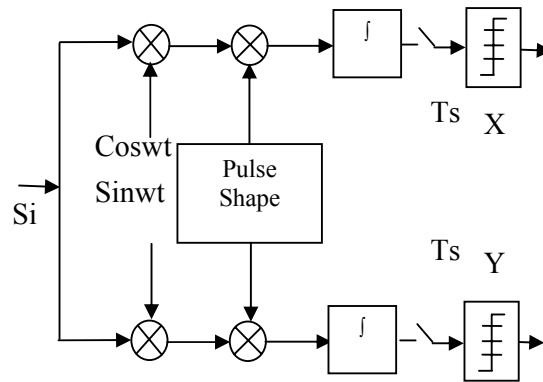


Fig. 3 Block Schematic of a QAM Demodulator

5. CONCLUSIONS

In this work a class of instructions namely Involved load Instruction in the form of fetch multiply and load, has been proposed for the VLIW architecture. The idea of Fetch Multiply and Load is different from the instructions used in a RISC processor. It is advantageous when there is continuous read from memory like those in a long FIR filter. The effectiveness of the involved load instruction in terms of speed increase has been demonstrated taking a QAM modem example. 15% reduction in number of clock cycles at an expense of approximately 2% increase in hardware has been reported at no expense in latency.

6. IMPORTANT REFERENCES

1. L.Petit and J.D.Legat "A low cost VLIW DSP Architecture for Communication equipment". URSI International symposium on Signals Systems and Electronics, 1998, pp 278-281.
2. J.L.Hennessy and D.Patterson Computer Architecture: A Quantitative approach, 2/e, Morgan Kaufmann Publishers, 1996.
3. TMS320C62X, CPU and Instruction set, Reference Guide, Texas Instruments, 1998.
4. L. Hanzo, W. Webb and T. Keller, Single and Multi-carrier QAM, John Wiley and Sons, Ltd.
5. Nat Seshan "High Velocity Processing", IEEE Signal processing magazine Mar-98.

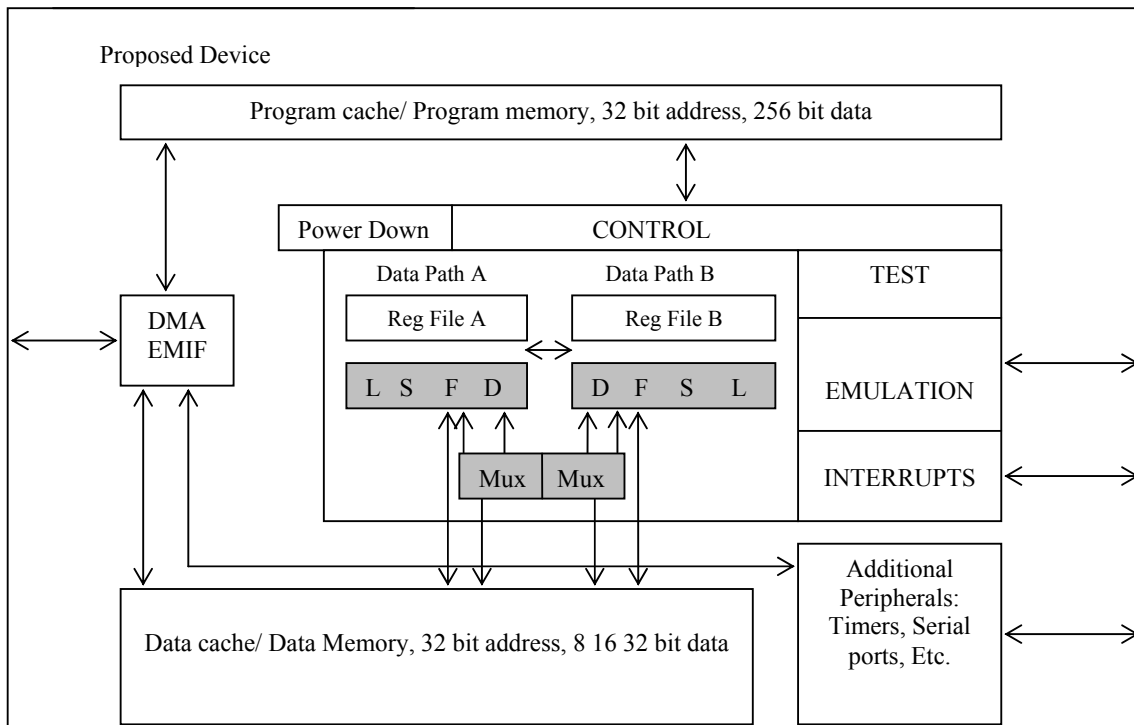


Fig. 4 The VLIW Architecture that is modified to include the FML instruction