A Multi-Thread Approach in Order to Avoid Pipeline Penalties

Gheorghe Ştefan
Politehnica University of Bucharest, Dept. of Electronics and Tc.
stefan@agni.arh.pub.ro

Abstract

The *simplest and smallest* pipeline structure is proposed to be used in the design of a multithread machine that runs without pipeline's penalties. The proposed structure implies two kinds of parallelism. The trivial solution is to see the resulting structure as a cluster of four processors. Another way, our way, is to find a specific architecture. The *master-slave architecture* proposes a hierachy of threads collaborating in running a computation. Some topics of application are suggested.

1 Introduction

One of the first machines which was able to execute many threads on the same physical structure is described in [Smith '78]. Many functional units and queues are put together in order to perform a multithread computation. This approach and others that follow it [Hirata '92] are characterized by the multiplication of resources and by their appropriate scheduling. A complex control mechanism is the main function that assures the system's performance.

The pipelined RISC approach offers a new perspective for a *small* and *simple* multithread machine. Thus, the **first goal** of this work is to propose a solution with minimal structural resources and simplest control.

The main idea is: *each stage in an usual pipeline structure can be used to perform a distinct step in the execution of an instruction belonging to a distinct thread of instructions.* If the successive instructions belong to distinct thread of instructions then all the penalties regarding data dependencies or branch execution will be avoided. Starting from this point of view, in [Ştefan '84], [Nikhil '89] and [Farrens '91] are presented solutions related with the proposal contained in this paper.

In [Ştefan '84] is presented a two-stage pipelined machine, in which two microprograms run, as two threads collaborating for implementing a LISP interpreter. The second thread was associated to a performant stack that supported the first thread, the main thread, that interpreted LISP programs. This approach introduces a bi-thread machine in which only the storage resources are multiplied, functional units remaining the same as in a single thread machine.

In [Nikhil '89] the Smith's approach is associated with a RISC pipeline machine. An additional queue, as in the Smith's machine, is needed to control the processing flow.

In order to avoid the dependencies between instructions in a RISC structure, Farrnes proposes the interleaving of a second stream of instructions in the pipeline, thus a machine with two threads free of hazards is obtained.

Nowadays there are more and more applications of the multi-thread philosophy. One of the most important is in data-flow applications [Nikhil '89], [Papadopoulos '91]. In parallel symbolic computation the multi-thread machine offers performant solutions [Halstead '88]. New concepts in operating systems need a multi-thread approach [Stallings '92].

The resulting structure can be used simply as a low-cost cluster of four processors tightly coupled. But,it deserves to look for a new architectural perspective offered by this specific structure. Thus, the **second goal** of this work is to offer a specific architectural support for this multi-thread structure. We believe and hope that the proposed *master-slave architecture* will be imposed as an efficient architectural style for the next trends in designing computing machines.

2 The Structure

D. A. Patterson answerd to the question "how to use 1000 registers?" [Sites '79] introducing the concept of *register windows*. An other answer can be: register windows and multi-thread execution on the same functional units [Ştefan '79, 84]. Now we are interested in promoting only the multi-thread exe-

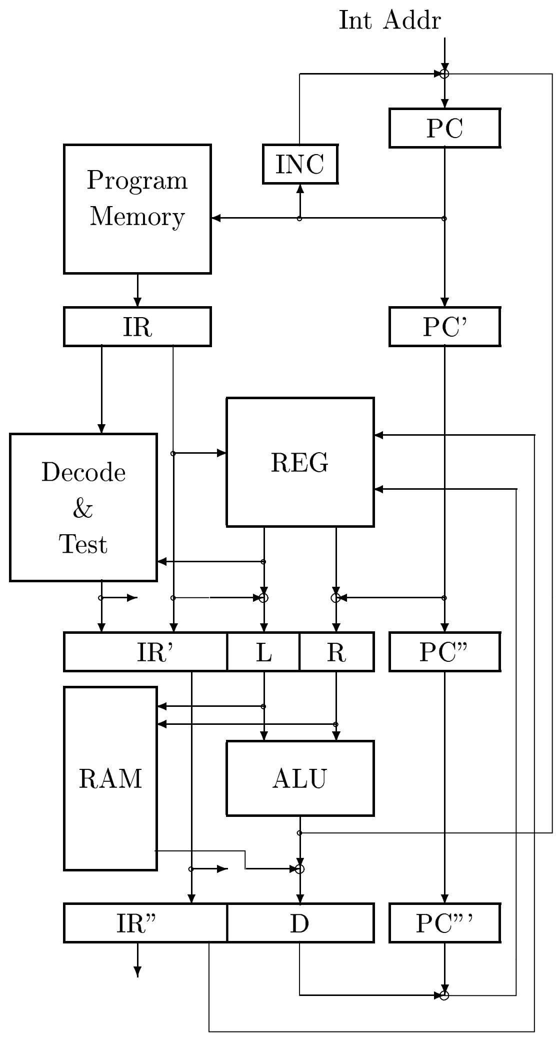

Figure 1: The smallest and simplest four-thread pipeline RISC machine.

cution on a single processor that has multiplied only the internal storage capability. Each thread has access to all registers and share the same functional units and the pipeline registers.

Let be the *simplest and smallest* pipeline structure of a RISC processor (see Figure 1). The simplicity of control and the minimal structure are allowed by the compilation and the cost is: a lot of "bubbles" in the pipeline execution. The content of the *program counter* (PC) must be delayed (with PC') one clock cycle in order to be synchronized with a value from the *instruction register* (IR). For saving the PC in registers (REG) the two registers, PC" and PC"', must be added. Our machine has an usual four-stage pipeline containing: **F**ETCH , **D**ECODE, **E**XECUTE, **W**RITE BACK.

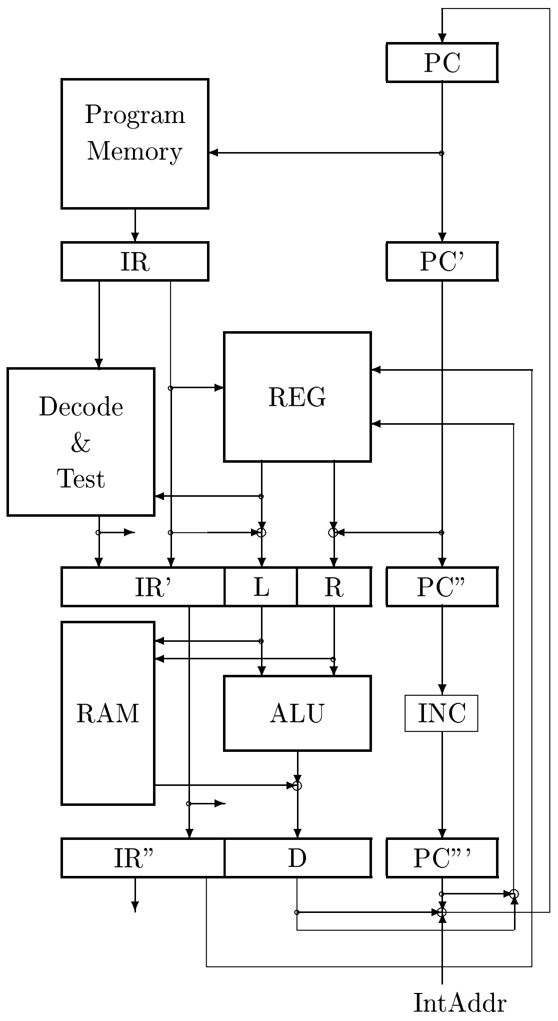

Figure 2: The multi-thread RISC structure: a low-cost cluster of four processors networkless interconnected

Starting from this pipeline structure a multi-thread machine is defined changing only the position of the incrementer (INC). The resulting processor (see Figure 2) performs in each *clock cycle* (T_i) on each stage an operation (O_i^j) for a distinct thread (the O action for the i-th instruction of the thread j). The following table shows the sequence of operations associated with the four threads.

T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
F_1^0	F_1^1	F_1^2	F_1^3	F_2^0	F_2^1	F_2^2	F_2^3	F_3^0	F_3^1
	D_1^0	D_1^1	D_1^2	D_1^3	D_2^0	D_2^1	D_2^2	D_2^3	D_3^0
		E_1^0	E_1^1	E_1^2	E_1^3	E_2^0	E_2^1	E_2^2	E_2^3
			W_1^0	W_1^1	W_1^2	W_1^3	W_2^0	W_2^1	W_2^2

Table 1: The interleaved execution.

This approach avoids the pipeline penalties, current in the single thread pipeline machines. Indeed, the successive operations on the pipe are uncorrelated, because they belong to the distinct thread of instructions. We obtain a cluster of tightly coupled four processors that share the same structural resources. The main problem to be solved is to associate to this structure an appropriate architecture.

3 Architectural Implications

We propose two distinct types of architectures to be associated to the four-processor cluster. The first supposes that the four threads are weak coupled and the second uses closely coupled threads.

3.1 A Cluster Architecture of Parallel Processors

The trivial architecture associated with the proposed structure is a MIMD parallel architecture. The four machines are physically connected without any interconnection path and share the same memory which can be used for communication. Each program runs without any pipeline penalties, thus the entire structure is maximally used in computation. Results a low-cost cluster. The processors communicate between them using messages stored in the shared memory.

The simplicity of the structure allows a physical implementation at a very high speed.

3.2 Master-Slave Architecture

The second architectural approach we propose is more sophisticated. It starts conceiving the processing as *computation* on data which must be *structured.* Any code associated with a program can be seen as a "coroutine" in which the control "jumps" between the two "threads":

- the main "thread" that performs the computation on one or more data structures
- the "co-thread" or the "co-threads" used for data structuring.

In the usual approach, the architecture of the computing system does not emphasize enough this distinction due to the fact that the von Neumann approach is centered on a single thread machine. But, if we have a good solution for a multi-thread structure, in which the communication between threads is performed by an efficient mechanism, then a special kind of architecture can be defined and implemented. We will call it: **master-slave architecture**.

A *master-slave architecture* consists in:

- a *master thread* that performs the main computation aided by
- one or more *slave threads* that manage one or more data structures implied in computation.

All the threads work in parallel and must communicate between them using a very performant mechanism.

There is a possibility that, for a good implementation and a lucky selected problem, the system performances grows more than n times, where n is the number of threads.

For example, in a Lisp Machine, the main thread computes the function EVAL and the co-threads are used for: stack management, lists management, garbage collector, compacting the free space. (A first experiment in this respect can be found in [Ştefan '84], where a single co-thread was used to implement the evaluation stack.)

3.3 The Cartezian Instruction Set

The first proposed *cluster architecture* does not need special communication instructions because the four thread are weak coupled and the communication time does not impose strong restrictions. But, in order to be efficient, the *master-slave architecture* needs a special approach in designing the communication mechanisms.

The master-slave architecture asks for a communication mechanism at the level of the instructions. For example, a slave-thread performing the stack function must receive a command from the master-thread in the machine cycle in which a PUSH or a POP is performed by the master-thread. In the same instruction the master-thread stores a number in the *top of stack* (TOS) and announces the slave to manage it. Therefore, we must design an instruction set which allows two synchronous commands. In the previous exemple the pair of commands is:

$$(storeinTOS)\&(PUSH).$$

The first is executed by the master-thread and the second is received by the slave-thread and executed with a minimal number of instructions.

Let us call this kind of instruction set **Cartezian Instruction Set**, because it presumes the cartesian product between two sets of instructions: a *main instruction set* and a *communication instruction set.*

We propose as the communication instruction set the following instructions, where $b = 0, \ldots, 7$ represents the bits of the first byte of the register $n + t$ from REG (see Figure 2) and $t = 0, \ldots, 3$ represents the thread's number.

SET b,t : set the bit b for the thread t

RST b,t : reset the bit b for the thread t

TSTZ b,t : **if** the bit b of the thread t is zero **then** performs the main instruction, **else** the main instruction is ignored and $PC \leftarrow PC$

TSTNZ b,t : **if** the bit b of the thread t is not zero **then** performs the main instruction, **else** the main instruction is ignored and $PC \leftarrow PC$

TZ&C b,t : **if** the bit b of the thread t is zero **then** complement it and performs the main instruction, **else** the main instruction is ignored and $PC \leftarrow PC$

TNZ&C b,t : **if** the bit b of the thread t is not zero **then** complement it and performs the main instruction, **else** the main instruction is ignored and $PC \leftarrow PC$

INT t : generates interrupt for the thread t

NOP : no operation.

The main instruction set, besides the standard RISC instructions contains two special instructions, devoted to the communication between the threads:

WCOM r : $PC \leftarrow (r)+PC$ (wait for a command)

CLD r,v : **if** $(r) = 0$, **then** $(r) \leftarrow v$, **else** $PC \leftarrow PC$ (conditioned load)

Of course, the main instruction set contains jumps to the address contained in an internal register (**JMP r**). This instruction, used in conjunction with instructions from the communication set allows a fast command of the slave-threads. For example, if the result of the addition of the content of the registers 2 and 7 must be pushed into a stack managed by a slave thread, then the next *chartezian instruction* must be performed by the master-thread:

ADD 2,7,28 TZ&C 5,2

where, the register 28 is TOS and the thread 2 is devoted to manage the stack. The bit 5 must be tested because the previous PUSH (triggered also by this bit) must be finished before the current PUSH is performed. The slave-thread "wait" with:

JMP (n+2)

where, the register $n+2$ contains the address where the instruction is stored having the bit 5 equal with 0. When the communication instruction on the master-thread complement the bit 5, the slave-thread starts to run the routine that executes the PUSH function (that reset the bit 5 at its end).

4 Applications

We mentioned already a trivial application: a cluster of four processors working as a MIMD machine. The main feature of this approach is given by the simplicity and by the loss of the network that interconnects the processors.

Another application can be a most sophisticated one: a Java Machine. The Java architecture is well suited for a multi-thread implementation. The main reason is the stack oriented architecture associated with the Java language. The proposed structure (see Figure 3) is a four-thread machine which has a suplemental loop closed over a microprogram ROM. The resulting structure can switch between two working operation modes:

- RISC-mode with the microprogram loop opened, $C/R = 0$ (see Figure 3)
- CISC-mode with the microprogram loop closed, $C/R = 1$.

Program Memory contains 8-bit words and *RAM* contains 32-bit words. *DCD/mP ROM* works as decoder in RISC-mode and in CISC-mode works as a microprogram memory. The *INC* circuit is enabled in RISC-mode and is disabled in CISC-mode. Any thread can switch independently between the two working modes.

The master-thread is devoted to run on the Java architecture. The slave-threads support the master-thread in managing, for example, the stack. Also, functions of the operating system can be performed by one of the slave-threads.

Program Memory contains some ROM modules for some fixed programs associated to the slave-threads. Thus, only the master-thread accesses programs stored in RAMs.

5 Conclusions

1. The pipeline multithread structure proposed by this paper is small and simple, well fitted for low-cost applications.
2. A cluster of five MIMD parallel machines tightly coupled is defined. These machines can share the same resources using a very fast communication mechanism. The interconnection network loss.
3. The concept of *Master-Slave Architecture* offers a large class of applications for our multithread structure. The distinction between the computation and the management of data structures is the main application of this kind of architecture.

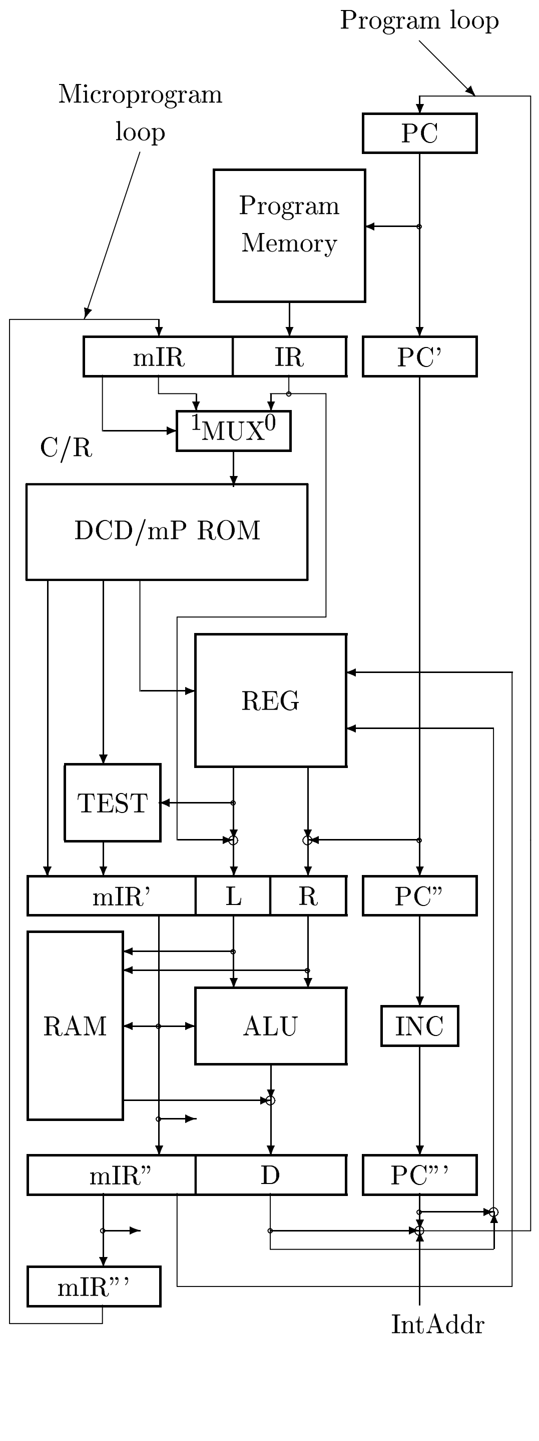

Figure 3: The organization of a **quadro C/RISC machine** with *interleaved multi-thread execution.*

4. The Cartesian Instruction Set and its implementation offers a very efficient mechanism for connecting all the four machines realized on the same physical structure. The communication is performed at the instruction level.

5. The symbolic computation is a very good application for the proposed machine because the degree of parallelism estimated for this domain is maximum five [Hwang '93].

References

[Farrens '91] M. K. Farrens, A. R. Pleszkun: "Strategies for achieving Improved Processor Throughput", *The 18th Annual International Symposium on Computer Architecture*, 1991. p. 362-369.

[Halstead '88] R.H. Halstead, Jr., T Fujita: "MASA: A Multithreaded Processor Architecture for Parallel Symbolic Computing", *The 15th Annual International Symposium on Computer Architecture*, 1988. p 443-451.

[Hennessy '81] J. Hennessy, et all: "MIPS: A VLSI Processor Architecture", *Proc. CMU Conf. on VLSI Systems and Computation*, Computer Science Press, 1981.

[Hennessy '90] J. Hennessy, D. A. Patterson: *Computer Architecture. A Quantitative Approach*, Morgan Kaufmann Pub., Inc., 1990.

[Hirata '92] H. Hirata, et all: "An Elementary Processor architecture with Simultaneous Instruction Issuing from Multiple Threads", *The 19th Annual International Symposium on Computer Architecture*, 1992. p. 136-145.

[Hwang '93] K. Hwang: *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, Mc. Graw-Hill, 1993.

[Jones '92] F. Jones: "A New Era of Fast Dynamic RAMs", in *IEEE Spectrum*, pp. 43-49, October 1992.

[Murakami '89] K. Murakami, et all: "SIMP (Single Instructiun stream/Multiple instruction Pipelining): A Novel High-Speed Single Processor architecture", *The 16th Annual International Symposium on Computer Architecture*, 1989. p. 78-85.

[Nikhil '89] R.S. Nikhil, Arvind: " Can Dataflow Subsume von Neumann Computing?", *The 16th Annual International Symposium on Computer Architecture*, 1989. p. 262-272.

[Papadopoulos '91] G. M. Papadopoulos, K. R. Traub: "Multithreading: A Revisionist View of Dataflow Architecture", *The 18th Annual International Symposium on Computer Architecture*, 1991. p. 342-351.

[Patterson '80] D. A. Patterson, D. R. Ditzel: "The Case of the Reduced Instruction Set Computer", *Computer Architecture News*, 8:6, (October), 1980.

[Patterson '85] D. A. Patterson: "Reduced Instruction Set Computer", *Communications of the ACM*, 28:1 (January), 1985. p. 8-21.

[Radin '82] G. Radin: "The 801 Minicomputer", *Proc. Symposium Architectural Support for Programming Languages and Operating Syatems*, Palo Alto, 1982.

[Sites '79] R. L. Sites: "How to use 1000 registers", *Caltech Conf. on VLSI*, January, 1979.

[Smith '78] B.G. Smith: "A Pipelined, Shared Resource MIMD Computer", *Proc. International Conference on Parallel Processing*, 1978. p. 6-8.

[Stallings '92] W. Stallings: *Operating Systems*, Macmillan Pub. Comp., 1992.

[Ştefan '79] G. Ştefan: *LSI Circuits for Processors*, Ph. D. Thesis in Polytechnical Institute of Bucharest, 1979. (in Roumanian)

[Ştefan '84] G. Ştefan, et all: "DIALISP - A LISP Machine", *The 1984 ACM Symposium on LISP and Functiunal Programming*, 1984. p. 123-128.

[Ştefan '96] G. Ştefan: *Five Stage Multithread Machine with Dual RISC Arhitecture*, research report, Center for New Electronic Architectures, 1996.

[Tabak '91] D. Tabak: *Advanced Microprocessors*, McGraw-Hill, Inc., 1991.

[Thistle '88] M. R. Thistle, B. J. Smith: "A Processor Architecture for Horizon", in *Proc. of 1988 Supercomputing Conf.*, pp. 35-41, November 1988.