A Flexible Register Access Control for Programmable Protocol Processors

Yutai Ma, Axel Jantsch and Hannu Tenhunen
Department of Electronics, Royal Institute of Technology, Sweden

Abstract- We investigate programmable processor architecture for internet protocol processing and use IP packet forward as a case study. We focus on flexible register access capability to support packet analysis and encapsulation. Compared to general-purpose superscalar microprocessors at least two-fold performance is achieved just with this improvement.

I. Introduction

Protocol processing has some distinctions from numerical and scientific appliations, such as bits and bytes operations, pattern matching, and high density of control operations, while numerical and scientific appliations are the targets of general-purpose microprocessors. On the other hand, communication protocols are ever evolving, the algorithms to implement these protocols are ever being improved, and many new services are emerging. All these indicate that programmable processors with flexible and powerfull protocol processing capability are needed.

Many fast table lookup algorithms have been proposed for IP routing. Three representives are [3, 1, 6]. [3] developed a method to compress the data structure so as to be able to make it fit into the secondary cache and thus to improve the overall performance. [2, 1] proposed an algorithm to improve the performance of IP routing by caching IP destination addresses/prefixes. The basic principle is that temporal locality exists in the internet traffic. [6] proposed a scalable IP routing algorithm based on binary quick search and hash functions, which is expected to work for IPv6.

This paper is devoted to programmable processor architecture for internet protocol processing. Programmability and high performance are our objectives. Our work indicates that programmable protocol processors can gain at least two-fold performance improvement over general-purpose superscalar microprocessors just with the new program control and flexible register access control.

II. Program Control

Branch penalty exists in general-purpose microprocessors. We noticed that conditional execution can not avoid branch hazard when the condition fails. Also, CASE statement is not supported directly by general-purpose microprocessors. Branch penalty may degrade the performance seriously in control-intensive protocol processing applications.

In [7] We proposed an FSM program control architecture. A porgram on that model is a sequence of control words as shown in Table 1. Each control word carries a base address to the next control word. The control field works with the branch field to generate the next address offset, as illustrated in Figure 1. We use two-bit address offset here to extend our work in [7] (one-bit offset), as shown in Figures 1 and 2.

We see that no branch penalty exists any more and CASE statement is supported directly by the new architecture. A segment/page register can be used to enlarge the instruction memory space. It should be stressed that the instructions can be pipelined.

Table 1: Control Word Format

Control Field	Branch Field	Next Instruction Base Address	Instruction Specification

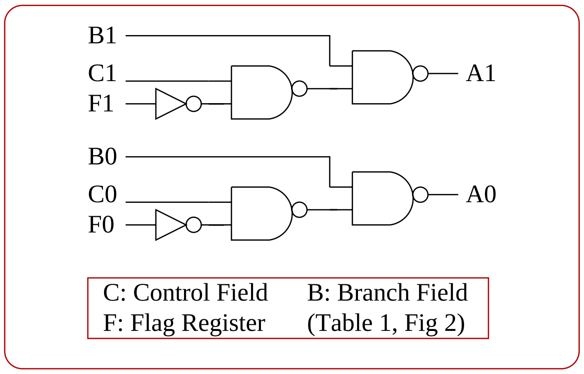

Fig. 1: Branch Target Address Offset Generation.

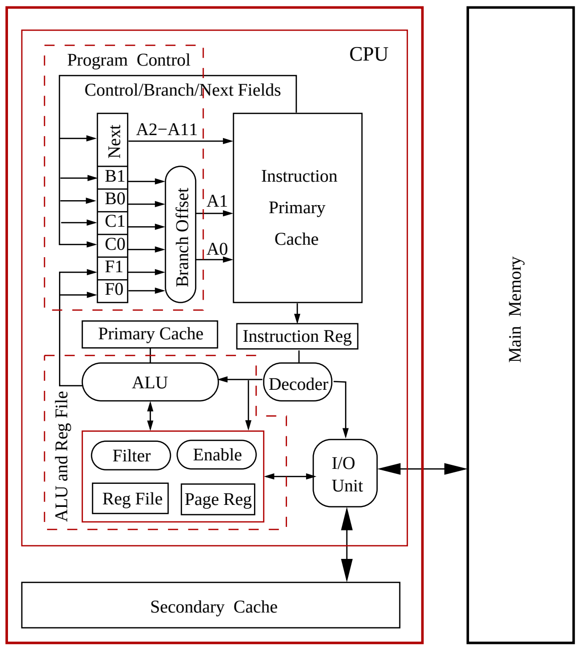

Fig. 2: A Programmable Protocol Processor (Program Control and Register Access Control).

Table 2: Register Encoding

5 Bits	1 Bit	2 Bits
Reg Index	Long/Subword/Byte (LSB)	Posi

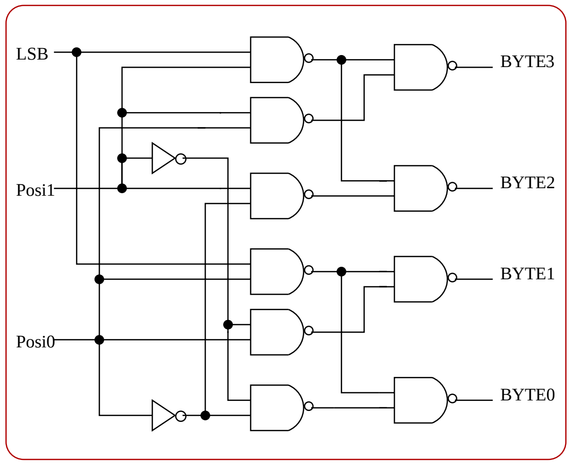

Fig. 3: Register Write Enable Logic and Register Read Filter logic (as shown in Figures 4–7).

III. Register Access Control

Because protocol processing involves a lot of bits, bytes, and subword operations, flexible register access capability has much effect on the system performance. For example, a 32-bit register may need to be referenced by its full 32-bit word, any 16-bit subword, or any byte. Such flexible register references are important to ease packet encapsulation and packet analysis.

A. Register Encoding

A simplified register encoding in our model is shown in Table 2. We classify register references as long word (4 bytes), subword (2 bytes), and byte. A long word is represented by LSB=1 and Posi=11. A subword is represented by LSB=1 with Posi=01 for 16 low-position bits and Posi=10 for 16 high-position bits, respectively. A byte is represented by LSB=0 with Posi=00, 01, 10 and 11 for the first, second, third, and the fourth byte respectively.

Two issues for implementing the flexible register access capability are how to control register read and write operations. Destination register write enable logic and source register read filter logic are shown in Figure 3. Becasue the register read/write enable signals are generated in parallel with register and instruction decoding, no time delay is contributed to the instruction cycle. The alignment between a source register/operand and a destination register will be described in conjunction with the instruction specifications of MOVE, AND/OR, and SHIFT.

B. Instruction Design

We take instructions MOVE, AND/OR and SHIFT as examples to show how to incorporate the flexible register access capability into instruction design.

A MOVE instruction format is shown in Table 3, where the "Count" specifies a left-shift count over the source register and it can be set by assembler or given by programmers. A MOVE instruction logic model is shown in Figure 4, where the filter masks out useless bytes. The alignment logic can be implemented by using a rotator with left-shift count of 0, 1, 2 or 3 bytes and thus this rotator is simple. The alignment contributes time delay to the MOVE instruction but this is not severe.

The MOVE instruction with an immediate operand is described in Table 4 and Figure 5.

Table 3: MOVE Instruction Format (1)

8 Bits	8 Bits	8 Bits	8 Bits
MOVE	Dest Reg	Source Reg	Count

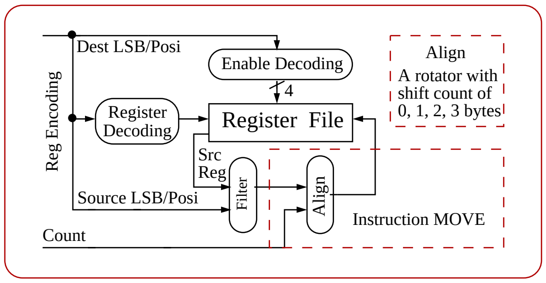

Fig. 4: MOVE Instruction Logic Model (1).

Table 4: MOVE Instruction Format (2)

8 Bits	8 Bits	16 Bits
MOVE	Destination Reg	Immediate Data

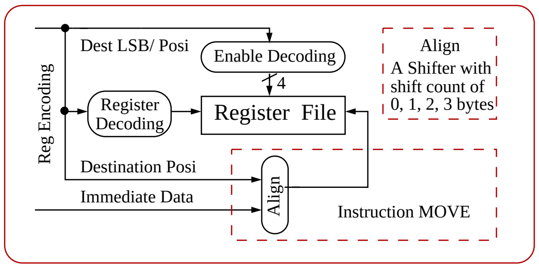

Fig. 5: MOVE Instruction Logic Model (2).

An AND/OR instruction format is shown in Table 5 and its logic model is shown in Figure 6. We take AND/OR with an immediate data as an example. To keep instruction length constant, we restrict the AND/OR operation with an immediate operand to be performed on the high/low 16-bit subword or on any byte. The AND/OR operation with a 32-bit immediate operand can be completed by using two such AND/OR instructions.

Table 5: AND/OR Instruction Format

8 Bits	8 Bits	16 Bits
AND/OR	Destination Reg	Immediate Data

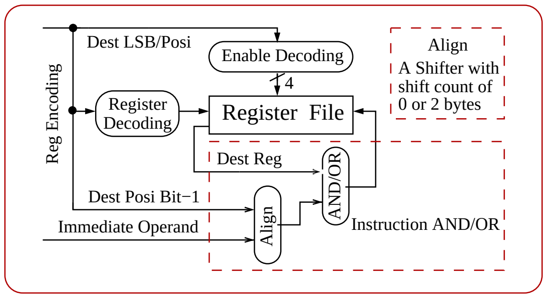

Fig. 6: AND/OR Instruction Logic Model.

A (left and right) SHIFT instruction format is shown in Table 6 and its logic model with an immediate shift-count is shown in Figure 7. This makes it easy to extract bits 12–15 or bits 8–11, for example.

Table 6: SHIFT Instruction Format

8 Bits	8 Bits	8 Bits	8 Bits
SHIFT	Dest Reg	Source Reg	Reg/Count

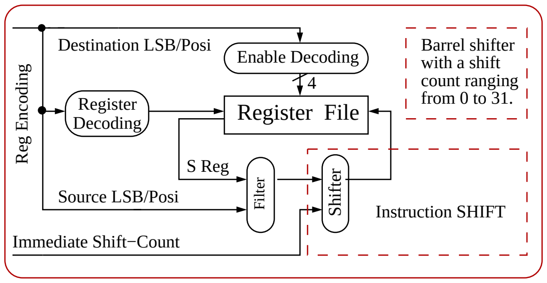

Fig. 7: SHIFT Instruction Logic Model.

We see that with the new MOVE, AND/OR and SHIFT instructions, any subword, any byte, and any binary string can be extracted out or set up easily.

IV. Case Study: IP Packet Forward

IP packet forward is to use destination address of an incoming packet to search for the next network address. To make efficient use of internet address space and to alleviate routing entries growth, classless interdomain routing (CIDR) protocol was deployed, by which the longest prefix match is performed each time. In this section we discuss efficient packet forward implementation on our new architecture and make a performance comparison with general-purpose superscalar microprocessors.

A. Compact Forwarding Tables

From [3] we see that it is possible to construct compact forwarding tables. Since all the data structures are located in primary or secondary cache, the overall system performance can be improved greatly. We believe that the scalable algorithm [6] can be also implemented efficiently on our architecture since it is based on the quick binary search algorithm.

Basically, we use Degermark's hierarchical IP forwarding tables [3] but with some improvements. The compact forwarding tables typically occupy 77–270 KBytes memory for the real-world IP routing.

B. Performance Comparisons

Performance comparisons of IP packet forward on our architecture and that in [3] are shown in Table 7. Three-way superscalar Pentium Pro and the implementation of [3] are chosen for fair and easy comparisons since the same algorithm is used for both implementations. Comparisons with other leading-edge protocol processors will be given in the future when more improvements are added to our architecture.

Look at prefix match in level-2 tables and assume the best case that all the referenced data structures are in on-chip primary cache [3]. In this case memory accesses consume 16 instruction cycles (8 memory accesses × 2 cycles/each) in [3] while the memory accesses in our model consume 24 instruction cycles (4 memory accesses × 6 cycles/each, assume all memory accesses are directed to the secondary cache). Taking this fact into account, 2.38 times (69/(37-8)) performance is achieved. When all the referenced data structures are in secondary cache in [3], the memory accesses consume 48 instruction cycles (8 memory accesses × 6 cycles/each). Taking this fact into account, 2.08 times ((101-24)/37) performance is achieved.

We see that our architecture achieves two-fold performance of the three-way superscalar Pentium Pro for this application. The performance improvement comes from two aspects: a new program control which leads to zero branch penalty and supports CASE statement, flexible register access capability which ease IP packet encapsulation and IP packet analysis.

Table 7: Performance Comparisons

Processor Model for IP Routing	Instruction Cycles		
	Level-1 Prefix	Level-2 Sparse	Level-2 Dense
Alpha 21164 (L1 cache)[3]	41	52–72, 100	75–83, 100
	148 when using L2 Cache		
Pentium Pro (L1 cache)[3]	25	36–40, 69	48–50, 69
	101 when using L2 Cache		
Max Expense Cache Access	8, L1 Cache	16, L1 Cache	
		48, L2 Cache	
New Architec	15	35, 36, 37	36
Max Expense Cache Access	12, L2 Cache	24, L2 Cache	

V. Conclusion

We investigate programmable processor architecture for high speed internet protocol processing. A penalty-free program control which supports CASE statement and flexible register access control which supports packet encapsulation and packet analysis are proposed. We see that two-fold performance is achieved compared to superscalar microprocessors based on the case study of IP packet forward.

We believe that the architecture can be further improved by applying other protocol processing oriented capabilities.

References

[1] Tzi-Cker Chiueh and Prashant, "Cache memory design for internet processors", IEEE Micro, january/February 2000.

[2] Tzi-Cker Chiueh and Prashant, "High-Performance IP routing table lookup using CPU caching", Proceedings of IEEE INFOCOM'99.

[3] Mikael Degermark, Andrej Brodnik, Svante Carlesson, and Stephen Pink, "Small forwarding tables for fast routing lookups", Proceedings of ACM SIGCOMM'97.

[4] V. Fuller, T. Li, and K.Varadhan, "Classless interdomain routing (CIDR): an address assignment and aggregation strategy", RFC-1519, September 1993.

[5] S. Keshav and Rosen Sharma, "Issues and trends in router design", IEEE Communications Magazine, May 1998.

[6] Marcel Waldvoge, George Varghese, Jon Turner, Bernhard Plattner, "Scalable high speed IP routing lookups", Proceedings of ACM SIGCOMM'97.

[7] Yutai Ma, Axel Jantsch and hannu Tenhunen, "A simple state transition control for FSM programmable protocol processors", Proceedings of IEEE MWSCAS'2000.