Instruction Set Architecture for Parallel Queue Processor

MARKOVSKIJ Arsenij,† ABDERAZEK Ben,†
SHIGETA Soichi,† HALCHAM Kutluk,† SOWA Masahiro†
and YOSHINAGA Tsutomu†

1. Introduction

In this paper we present the design and analysis of *Instruction Set Architecture* (ISA) for *Parallel Queue Processor* (PQP). It is the most critical stage during the design of a new architecture. Computer architect must clearly understand consequences of decisions made in ISA for the whole success of research. ISA strongly affects all aspects of system: compilation time, code size, hardware complexity and performance, etc. Program size is an important concern in many applications, especially for those requiring small memory footprints. The Queue-based instruction set is a promising approach for reducing code size and system complexity. Our Instruction Set consists of fixed-length 2-byte instructions. First, we explain formats used in our instruction set. Then we make preliminary evaluation of PQP program size through a set of simple benchmarks.

Parallel Queue Processor is based on Queue Computational Model (QCM) which uses Queue (FIFO memory) instead of registers as an intermediate storage of operands. This allows significant reduction in ALU instruction length because we do not need to specify register names — location of operands and results is *implicit*. Unfortunately, there are other instruction classes such as Load/Store and Branches that can not be shortened so easily. In such case variable-length ISA would lead to the most compact code. On the other hand variable-length instructions complicate Instruction Decode Unit of CPU. As a tradeoff between code-size and hardware complexity we choose 2-byte fixed-length ISA.

† Graduate School of Information Systems, University of Electro-Communications, Tokyo

2. Instruction Set Architecture

Main types of instructions are:

- ALU (add/sub, logical, shift, mul/div)
- Load/Store
- move from/to Queue to/from Registers
- branch

2.1 ALU

ALU-type instructions include perform addition/subtraction, logical, shift, multiplication/division operations. Their format is given on **Fig. 1(a)**. Most Significant Byte (bits 15-8) specifies opcode. Bit 7 (called *p-bit*) selects one of 2 ALU addressing modes:

- *Produced-Consumed Order* addressing mode is used when p-bit is cleared. On **Fig. 2(a)** `ADD` instruction "consumes" 2 operands from the *Head* of Queue marked by *QH* (Queue Head) pointer, performs computation, and writes back ("produces") the result at the *Tail* of Queue which is marked by *QT* (Queue Tail) pointer. Bits 6-0 are left unused.
- *Produced Order* addressing mode (p-bit is set) gives a freedom to choose the second operand by specifying its *offset* (bits 6-0)

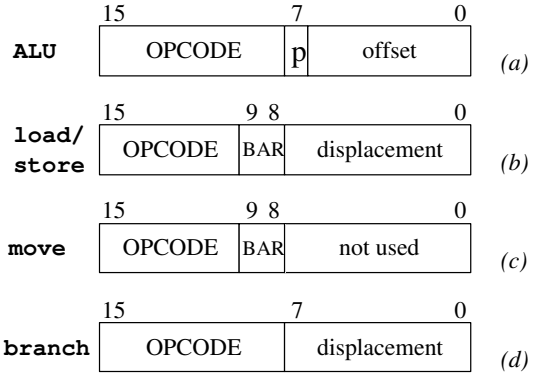

Fig. 1 Major instruction formats

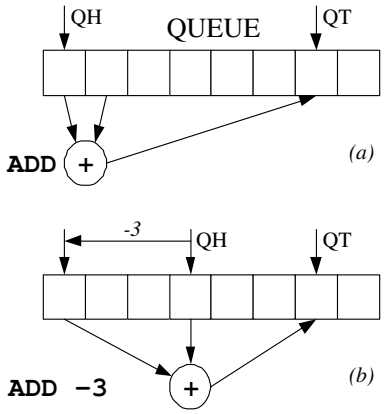

Fig. 2 Produced-Consumed *(a)* and Produced Order *(b)* ALU addressing modes

from QH. On Fig. 2(b) the second operand of `ADD -3` instruction is found at $QH - 3$. This approach allows us to *reuse* data that was "consumed" previously.

2.2 Load/Store

PQP is *load-store* machine. To access data memory we use *Displacement* addressing mode where *Effective Address* (EA) of memory is sum of *displacement* and *Base Address* (BA):

$EA = BA + displacement$

PQP has 4 *Base Address Registers* (BAR) to store Base Address. Format of load/store instructions is given on Fig. 1(b). Bits 15-10 specify opcode, bits 9-8 select particular BAR, and Least Significant Byte (bits 7-0) is displacement. In cases when 1-byte is not enough[6] displacement may be *extended* till 2 bytes by mean of special instruction called `COVOP`.

2.3 Move

These instructions (see Fig. 1(c)) move data between Queue and BAR. Bits 9-8 specify particular BAR, and Least Significant Byte is left unused.

2.4 Branch

Branches (see Fig. 1(d)) are *PC-relative — Target Address* (TA) is a sum of *displacement* (bits 7-0) and *Program Counter* (PC):

$TA = PC + displacement$

3. PQP simulator

We wrote *PQP Simulator* using *Verilog HDL* programming language. The size of simulator is about 2,000 lines. It outputs the following information:

- very detailed information about instruction set usage like instruction distribution in the program.
- run-time program size
- program execution time (in clock cycles)
- maximum and average *Instruction Level Parallelism* (ILP)

Table 1 Program size of PQP vs. MIPS

benchmark name	FFT	Fibonacci
program size reduction (%)	35	37

We wrote several simple programs for PQP and MIPS instructions sets and compared corresponding code sizes. Preliminary results are given in **Table 1**. At this moment we are concentrated on writing more extensive benchmarks.

4. Conclusions

In this paper we presented the design of Instruction Set Architecture for Parallel Queue Processor. We also wrote PQP Simulator using Verilog HDL programming language. Preliminary simulation results show that PQP program size is about 35% smaller than corresponding MIPS code.

References

1) B. R. Preiss, "Data flow on a queue machine", PhD thesis, Department of Electrical Engineering, University of Toronto, 1987.
2) S. Okamoto, A. Maeda, M. Sowa, "Superscalar processor based on queue machine computation model", IPSJ SIG Notes, vol.98, no.37, ARC–129–3, pp.13–18, May 1998.
3) S. Okamoto, H. Suzuki, A. Maeda, M. Sowa, "Design of a superscalar processor based on queue machine computation model", IEEE PACRIM'99, pp.151–154, August 1999.
4) M. Sowa, "The foundation of parallel queue machine", Technical Report SLL–00330, Distributed Processing Laboratory, Graduate School of Information Systems, University of Electro-Communications, Tokyo, 2000.
5) A. Markovskij, "Producer-order parallel queue processor architecture design", Master thesis, Graduate School of Information Systems, University of Electro-Communications, Tokyo, 2002.
6) J. L. Hennessy, D. A. Patterson, Computer Architecture A Quantitative Approach, Morgan Kaufmann Publishers, San Francisco, California, 2003.