社団法人 情報処理学会 研究報告
IPSJ SIG Technical Report

2003－HPC－94 (4)
2003／6／13

A Reduced Bit-Width Instruction Set Architecture for FQM Execution in Hybrid Processor Architecture (FaRM-rq)

BEN A. ABDERAZEK ,† SOICHI SHIGETA ,† TSUTOMU YOSHINAGA †
and MASAHIRO SOWA †

Code size is a critical concern in many applications, especially for those requiring small code size and special cores. The Queue based instruction set is a promising approach for reducing code size and system complexity.

In this paper, we present an efficient narrow space instruction set architecture for a Queue mode execution (FQM) in a functional assignment register microprocessor that supports a multi instruction sets through run time functional assignment. In FQM mode, the system executes queue based instruction set (termed rwQIS) that are carefully designed with a limited opcode and access to a limited set of special registers. The rwQIS is targeted for a low system complexity and reduced Bit-Width Instructions.

In addition to the instruction set architecture, we give a measure of the expressive power of FQM instruction set by the relative density and the code ratio of some benchmark programs.

1. Introduction

One of the primarily goals of a computer architect is the design and construction of machines that support the efficient execution of the programs that will run on them. The simplicity of the instruction set provides a number of implementation advantages that can substantially enhance the performance of the machine. For example, the restriction that arithmetic and logical operation's operand be located in an internal processor storage may permit the number of pipeline stages and/or their duration to be reduced resulting in faster execution of a given application. Also the use of fixed length instructions and a few formats permits simpler hardware and faster instruction decoding.

Recently, various design for Queue processors have been proposed, and some hardware designs are currently under constructions. The work presented in[3),4),6)] presents examples of research in progress or already finished. In any stored-program in these architectures, information is constantly transferred between the memory and the instruction processor. Since transfer bandwidth is a limited resource, inefficient in the encoding of instruction information can have definite hardware and performance cost. Such and other considerations supported the development of a so named produced order parallel Queue processor (PQP)[1),2),6)]. The earlier version of the PQP system provides variable instruction length and relatively compact encoding of computations and, therefore, the use of fixed length instructions with few formats was not the target of the earlier prototype. As a result, the performance benefits of the PQP architecture was offset by higher fetch alignments and instruction decode overhead. In addition, it has been conjectured that the resulting variable instruction length and the large number of instruction formats could harmfully affect memory bandwidth performance[4)]. Thus, to keep balance between hardware simplicity and overall performance, we have decided to enhance the earlier version by designing a fixed instruction set architecture that will be supported by FQM mode within the hybrid FaRM architecture. One major reason for this strategy is that decoding instructions of a simple fixed-length format can be done in a single processing step. However, with variable-length instruction format, decoding is context-sensitive; determining an instruction's length and, therefore, the start of the following instruction and the position of instruction fields, is a multiple-step process that requires examining some instruction bit values to determine how to extract and interpret the rest of the instruction fields. This can critically complicate the design of our proposed architecture. Another important concern is the density consideration. The strategy we used for increasing density is to design a sixteen-bit fixed-length format. By keeping all the instructions

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

the same length, fetch and decode simplicity can be maintained. However, the short instructions may limits the number of referenced queue entries. Another problem is the shortage of memory offset addressing range. We will show that these limitations are avoided by adopting compiler and hardware techniques.
The main consideration of this paper will be the description of the rwQIS architecture and facilities that have been developed to contribute to a high degree of performance and simplicity of the FQM execution.
The rest of this paper is organized as follow: In section two, we first present a brief overview of the earlier and the rwQIS instructions features. Section three, gives the rwQIS architecture design details and facilities that have been developed. We give the performance evaluation of the rwQIS and we compare it to the earlier designed version in section four. In the last section we give concluding remarks and future work.

2. Instruction Set Classifications

PQP and PQPpf (parallel queue processor with produced order scheme and fixed instruction set) are both based on queue computation model. They use a FIFO data structure as the underlying mechanism for results and operand manipulations; that is, each instruction removes the required number of operands from the front of an operand Queue (OPQ), performs some computation and stores the result back to the operand Queue. PQP and PQPpf are nearly identical in function and supported on the same pipeline with identical execution unit resources. Both have the normal complement of ALU, shift, memory, and floating point operations. The principle differences lie in the size and format of instructions encoding and the extra registers set. PQP instructions are 8 and 24 bit length. While PQPpf instructions are sixteen-bit fixed length. Because the information density of 16-bit instructions is higher, a more elaborate encoding scheme is necessary. Both instruction set define Queue execution model, though internally they are register machine-like. PQPpf has operation with implicit operand and/or destination and also has some operations that operate on register. PQPpv and PQPpf instructions are both executed on five stages execution pipelines.

3. rwQIS Instruction Set Architecture

The rwQIS instruction set is intended to be used in Queue mode execution (FQM) within FaRM processor. All instructions are byte addressed and provide access for bytes, half words, words, and double words. Bellow we will discuss the instruction set design considerations and the facilities that have been developed for FQM mode.

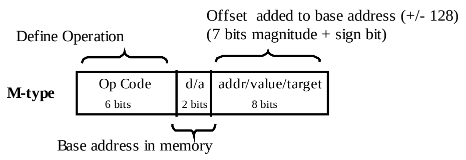

Fig. 1 Memory instructions (M-type) format

3.1 Memory Type Instructions

The memory type (M-type), shown in Fig. 1, consists of all load and store operations. When data must be obtained from/sent to memory, the M-type instructions are needed. The *op* field is 6-bit and is used for operation coding. The *d* field is 2-bit and is used to select one of four data registers. The *addr* field is 8-bit offset to be added to the base address (in one of the *d* registers).
For load instructions (i.e., lw) the content of one of the the *d* registers is added to the 8-bit offset to form the 32-bit address of the memory word. The word is loaded from memory to a Queue entry pointed by the Queue tail (QT) pointer. In Fig. 2 (b), the memory instruction would be decoded as *load the 8 bit byte at memory location [contents of d0] + 0x52 into the Queue tail* pointed by the Queue tail (QT) pointer

The store instruction has exactly the same format as load, and use the same memory calculation method to form memory addresses. However, for store instructions the data to be stored are found from the head of the operand Queue (OPQ) indicated by the Queue head (QH) pointer. In Fig. 2(a), the memory instruction would be decoded as *store the 32 bit word of the OPQ entry indicated by the QH at memory location [contents of d1] + 0x53.*
Memory Address Extension: In M-type instructions, the offset is only 8-bits wide; that is the address space range (from the base address) is only 128 memory slots. This may not be large enough for real applications. To cope with this address "shortage", we adopted the idea proposed by Sowa[5]. In the above idea, the com-

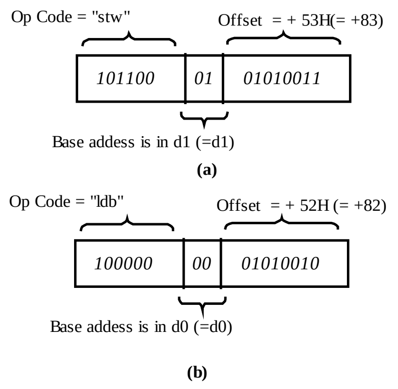

Fig. 2 Load and Store instructions internal coding example

piler uses static optimizations techniques and automatically inserts (when needed) a convey instruction before each load or store instruction. The convey instruction is simply an instruction which forwards its operand (offset) to the consecutive load or store offset field. That is, when a convey instruction is inserted before a load or store instruction, the processor combines the convey instruction offset with the current load or store instruction offset and the data register to find the effective address. The convey instruction utilization is illustrated in Fig. 3.

```
; address space extension
covop 25H       ; convey 25H address (offset)
ldb     12H(d0) ; load data from mem[25H(12H(d0)]
..
..
..
..
addp  4            ;
stw   24H(d1) ; store word at mem(24h(d1))
```

Fig. 3 Address space extension

3.2 Data/Address Register Instructions

The instruction set are designed with four data registers (d0∼d3) and four address (a0∼a4)) registers. These registers are used as base addresses for memory and control instructions respectively. The control instructions, which will be described later, consist of jump, loop, call, and interrupt instructions. The data/address registers are general purpose; that is they are visible to the programmer. These registers are 32-bits wide. Therefore, to set or reset one 32-bits address register, four instructions (sethh, sethl, setlh, setll) are needed. It may seam that the it set/reset operations are costly since four instructions are needed to set one address or data register. However, from our preliminary evaluations, operations on these registers occur not so often within a give application. Figure 4 is an example showing how data register *d0* is set with *setxx* instructions. Note that these instructions have the same format as M-type instructions.

The data/address registers can be also in-

```
; set data  register d0 with address A4121B45H
sethh  d0, A4H   ; set bits 23~31 of register d0
sethl  d0, 12H   ; set bits 16~24 of register d0
setlh  d0, 1BH   ; set bits 8~15 of register d0
setll  d0, 45H   ; set bits 0~7 t of register d0
; address space extension
covop 25H        ; convey 25H address (offset)
ldb      12H(d0) ; load data from mem[25H(12H(d0)]
..
..
addp  4          ;
strw   24H(d1) ; store word at mem(24h(d1))
```

Fig. 4 Address register setting example

cremented or decremented by the *inc* instruction. The syntax is: *inc a, value.* This type of instructions belongs to the I-type instructions shown in Fig. 5. Note that the range

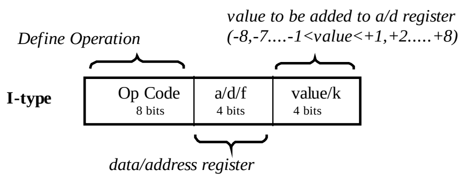

Fig. 5 I-type Instruction Format

of the *value* operand is:$-8, -7, \cdots, <$ *value* ¡ $+1, +2, \cdots, +8$. The I-type instruction also consists of *swi* (software interrupt) and *setr* (set register) instructions.

3.3 Control Instructions (C-type)

The control instructions consist of *move, branch, jump, loop, call,interrupt, and barrier* instructions.

The *jump,loop and call* instructions have the same format as the previously defined M-type instructions. They all use *a* register as a base address register and an offset target of eight bits. As with memory instructions, target addresses of these instructions can be extended to sixteen bits by *convey* instruction. The C-type shown in Fig. 6 also has other control instruction with only one operand.

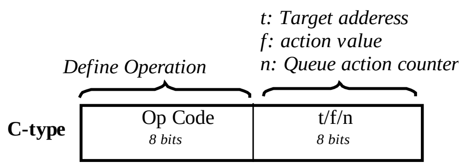

Fig. 6 C-type Instruction Format

3.4 Transfer Instructions

The FQM supports four types of control flow (transfer) change: (1) Barrier-Branch, (2) Jump, (3) Procedure call and (4) Procedure return. As illustrated in Fig. 6, the target address (t) of these instructions is always explicitly specified in the instruction. Because the explicit target (displacement) value, which will be added to the fetch counter to find the real target address (RTA), is only 8-bits the *convey* instruction's offset can be combined with the transfer instructions' explicit target to extend the RTA space.

Branch Instruction: The branch instructions belong to the C-type. To avoid having too much work per instruction, the branch instruction resolution is divided into tasks (1) whether the branch in taken (with comparison instruction) and (2) the branch target address (address calculation). One of the most noticeable properties of the FQM branches is that a large number of the comparisons are simple tests, and a large number are comparison with zero. According to the type of the condition the comparison instruction compares two entries obtained from the head of the OPQ and insert the result (true/false) to a condition code (CC), which is automatically checked by the branch instruction. In our implementation, branches are also barrier instructions. That is, all instruction preceding the branch instructions should complete execution before new instructions (branch successor instructions) can be issued.

Barrier Instructions: This type consists of *halt, barrier, SerialOn, and SerialOff* instructions. These instructions are designed to control the execution and the process type of instructions.

Queue Control Instruction (QCI): The QCI consists of *stpqh* (stop Queue head), *stplgh* (stop life Queue head), *autqh* (automatic Queue head), and *autlqh* (automatic life Queue head). These instruction are designed to control the life of data within the (OPQ).

3.5 Producer Order Instructions

This type (P-type) consists of about 70% of the total instructions in FQM execution. It consists of all single and double word computing, logical, compare, and conversion instructions. The format of the P-type instruction is illustrated in Fig. 7. We have to note that both integer and floating-point operations are supported.

Table 1 FQM Instructions Distribution

Class Type	Inst. Nbr.	Percentage
M-type	31	21.67
I-type	11	7.67
C-type	24	16.78
P-type	61	42.65
Others	16	11.23

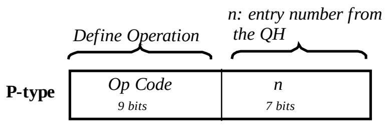

Fig. 7 P-type Instruction Format

4. FQM Instruction Set Performance

The measure we used to estimate the performance of the expressive power of the FQM instruction set is: (1) the path length, the total number of instructions in an execution trace, and (2) the relative density of two programs that encode the same computation and is the ratio of their sizes.

We have to note that the FQM model attempts to speedup execution my three means: (1) reduces the path-length (instruction count) by using a more powerful instruction set. (2) Eliminates the instruction extraction stage and avoids the introduction of other delays since all instructions are fixed length and no need to align them as with the previous PQPpv processor and (3) Reasonable Relative code density.

4.1 FQM Instructions Classifications

The rwQIS instructions are classified in Table 1. The P-type instruction occupies nearly half of the total instruction number. However, the I-type instructions consists of the smallest percentage of the total instructions.

4.2 Code Density

To evaluate the code density, we compared some benchmark programs code size for variable instruction set with the called rwQIS instruction set. The comparison result is shown in Fig. 8. From the above figure, we can see that the code density for programs with fixed length instruction set (rwQIS) is considerably

lower that programs with variable instruction set. In average, the code density for rwQIS programs is about 2.9 less that variable width instructions programs.
For some programs, we had relatively low expectations for the rwQIS (16-bit fixed size) instructions Vis a Vis earlier designed variable version. We expected that the weaker instructions set would increase the path length for programs, and this increase would preclude serious interest in the instruction set for a practical machine. This assumption proved wrong for two reasons; first the path length penalty was not as great as we anticipated, and second, there are performance limits that are independent of instruction scale and packaging that 16-bit instruction set desirable.

4.3 Density Ratio and Path Length

Another measure is of the expressive power of an instruction set is the relative density of two programs that encode the same computation is the ratio of their sizes. The size ratio is shown in Fig.9.

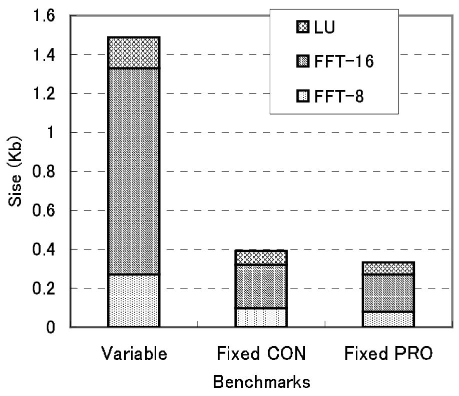

Fig. 8 Code size figure

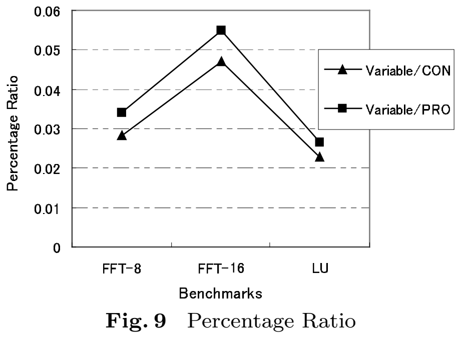

Fig. 9 Percentage Ratio

5. Conclusion and Future Work

In this article, we presented an efficient narrow space instruction set architecture for a Queue mode execution (FQM) in a functional assignment register microprocessor that supports a multi instruction sets through run time functional assignment. The rwQIS is targeted for a low system complexity and reduced Bit-Width Instructions.
We measured the expressive power and performance of FQM instruction set by the code size ratio and the code density of some programs that encode the same computation. We conclude that the prosed rwQIS set leads to efficient code density and is expected be implemented wit simple decoding circuitry when compared with the earlier designed version

6. acknowledgements

This work is partly supported by JSPS Grants-in-Aid for Scientific Research (C), No.15500033.

References

1) Abderazek B. A., Kirilka N., Sowa M.: FARM-Queue Mode: On a Practical Queue Execution model. Proc. of the Int. Conf. on Circuits and Systems, Computers and Communications, Tokushima, (2001) 939-944
2) Sowa Lab.: http://www.sowa.is.uec.ac.jp
3) Suzuki H., Shusuke O., Maeda A., Sowa M.: Implementation and evaluation of a Superscalar Processor Based on Queue Machine Computation Model. IPSJ SIG, Vol.99, N0.21,pp. 91-96 (1999).
4) Abderazek B.A.: Dynamic Instructions Issue Algorithm and a Queue Execution Model Toward the Design of a Hybrid Processor Architecture. PhD. Thesis, IS Graduate School, Univ. of Electro-Communications, (2002).
5) Sowa M.: Fundamental of Queue machine. The Univ. of Electro-Communications, Sowa Laboratory, Technical Reports SLL30305, (2003).
6) Sowa M., Abderazek B.A, Shigeta S., Nikolova K., d Yoshinaga T.: Proposal and Design of a Parallel Queue Processor Architecture (PQP), 14th IASTED Int. Conf. on Parallel and Distributed Computing and System, Cambridge, USA,pp.554-560 (2002).