

# Subword Permutations with MIX Instructions

Zhijie Jerry Shi

Department of Computer Science and Engineering, the University of Connecticut, Storrs, CT 06286 USA

Email: zshi@engr.uconn.edu

**Abstract-** Subwords are data items smaller than a word. Subword parallelism, a technique that packs multiple subwords in a register and processes them in parallel, has been used to accelerate multimedia applications significantly. However, subword computations also raise new challenges, one of which is the subword permutation problem, i.e., efficient rearrangement of subwords within a register or among several registers. MIX instruction is one of several instructions that have been implemented in general-purpose processors to address the subword permutation problem. The MIX instruction is very effective to perform certain subword permutations, namely, subword matrix transposes. However, it is still not clear whether MIX instructions can perform other types of permutations efficiently and, if they can, what subword permutations can be performed with what MIX instructions. This paper tries to answer these questions. It first gives a more general definition of MIX operations and then describes a class of subword permutations that the MIX operations can achieve. It also examines the instruction set architecture (ISA) support for the generalized MIX operations.

## I. INTRODUCTION

Traditionally, general-purpose processors are optimized for word-oriented computation. They considered all the bits in a register as one unit, a word. Hence, their instruction set architecture (ISA) provides limited support for the manipulation of data items smaller than a word. Few instructions that operate at the bit level are logical (AND, OR, XOR, and NOT) and SHIFT operations.

Recently, multimedia instructions have been added into many processors' ISA to accelerate multimedia processing. For example, MAX-2 is the multimedia extension in PA-RISC 2.0 [1] [2], MMX, SSE, and SSE-2 are in IA-32 [3], and AltiVec is in PowerPC [4]. These multimedia extensions are based on the same idea: subword parallelism. Subwords are data items smaller than a word. Multiple subwords can be packed into one register and be processed in parallel. In subword parallelism, subwords need to be rearranged efficiently within a register or among multiple registers, so they can be placed at proper positions for parallel processing. Without such fast arrangement of subwords, called *subword permutation*, the performance gain achieved by subword parallelism may diminish.

Several instructions have already been defined to address the subword permutation problem. For example, PERMUTE and MIX are two instructions implemented in MAX-2 [2]. The PERMUTE instruction can perform any arbitrary permutation of 16-bit subwords stored in one source register. For subwords in multiple registers, Lee invented the MIX instructions to combine even or odd subwords from two

source registers [6]. While these are the first subword permutation primitives introduced into general-purpose microprocessors, both PERMUTE and MIX dealt only with 16-bit subwords in MAX-2 [1] [2]. IA-64 extended MIX to support 8-bit subwords and added five variants of byte permutation instructions called MUX [5].

Lee further defined new subword permutation primitives and proposed a minimal canonical instruction set for 2-dimensional rearrangements of subwords packed into registers [7]. The minimal canonical set includes MIX and PERMSET, and both are defined for subword sizes that are powers of two. Lee showed that the minimal set can efficiently perform any transforms of a 2x2 matrix. However, the question of how the minimal set can perform arbitrary transforms of larger matrices is still open.

Recently, several bit permutation instructions have been proposed to perform arbitrary bit-level permutations for bits and subwords stored in one register. The bit permutation instructions include GRP [8], OMFLIP [9], CROSS [10], [11], and BFLY[12]. They are able to permute bits or subwords of multiple bits efficiently, requiring  $\log_2 k$  instructions for  $k$  bits or subwords. But they also require all the bits or subwords be stored in one register. Shi proposed a method that uses bit permutation and MIX instructions to perform arbitrary permutations of up to  $n$  subwords, where  $n$  is the number of bits in a register [13]. The  $n$  subwords can be of multiple bits and stored in more than one register. The method, however, can not be applied to more than  $n$  subwords.

We noticed that although MIX is one of the first subword permutation instructions, it is useful and effective in many applications such as matrix transpose in JPEG and MPEG, subword sorting [14], and subword permutation across multiple registers [13]. However, to our best knowledge, it has not been studied which subword permutations can be performed efficiently with MIX instructions, and what MIX instructions and how many of them are needed for these subword permutations. In this paper, we try to answer these questions.

The rest of the paper is organized as follows. In Section II, we describe the MIX instructions implemented in MAX-2 and IA-64. Section III gives a more general definition of MIX operations and a class of permutations they can perform. Section IV investigates the ISA support for the generalized MIX operations and Section V concludes the paper.

## II. MIX INSTRUCTIONS

The MIX instruction, first defined in MAX-2 [1], has two versions, MIX left and MIX right. Both of them take two

source registers and produce one destination. The subwords in the source registers are divided into pairs. A MIX left instruction takes the left subword in each pair, alternatively from the two source registers while a MIX right instruction takes the right subwords. Fig. 1 illustrates the MIX instructions defined in MAX-2. In the figure, R1 and R2 are two source registers containing four subwords each. The MIX left instruction alternatively takes the left subwords from R1 and R2, and stores the result in R3. The MIX right instruction takes the right subwords and stores them in R4.

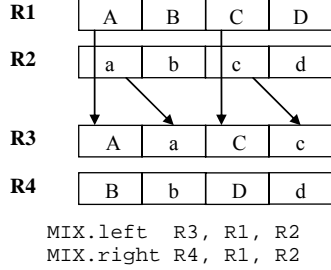


Fig. 1: MIX instructions in MAX-2

Since MAX-2 is defined on 64-bit registers and supports only 16-bit subwords, each source register of MIX instruction has four subwords. IA-64 added the support for 8-bit subwords, increasing the number of subwords in a register to eight [5].

The subwords in Fig. 1 form two 2x2 matrices. A, B, a, and b are in one matrix and C, D, c, and d in the other. The two MIX instructions actually transpose the two 2x2 matrices in parallel. MIX instructions can also be used to transpose larger matrices [1] [13] [14].

### III. MIX OPERATION

In this section, we will first give a more general definition of MIX operation, and then study its properties.

#### A. MIX operation

As described in Section II, the MIX instruction has two variants, MIX left and MIX right. When both instructions are used, every subword in the source registers appears once and only once in the destination registers. So with a pair of MIX left and MIX right instructions, we actually perform a permutation on the subwords in the two source registers. In Fig. 1, for example, the pair of MIX instructions performed a permutation on eight subwords: (A, B, C, D, a, b, c, d)  $\rightarrow$  (A, a, C, c, B, b, D, d).

The MIX instructions can be defined for different subword sizes and result in different permutations. In this paper, we express the subword sizes in terms of *minimal subwords*, which are the smallest data items we need to process. Depending on applications, the minimal subword can be one bit, eight bits, or other sizes. A subword consists of one or more minimal subwords. We consider only subwords that consist of  $k$  minimal subwords, where  $k$  is a power of two. For convenience, subword in this paper is used to refer to all data items, even if some of them may be larger than a word.

Suppose a sequence  $S$  of  $m$  subwords is  $(w_0, w_1, w_2, \dots, w_{m-2}, w_{m-1})$  where  $m$  is even. A permutation  $P_m$  on  $S$  is defined as  $P_m(S) = (w_0, w_{m/2}, w_2, w_{m/2+2}, w_4, w_{m/2+4}, \dots, w_{m/2-2}, w_{m-2}, w_1, w_{m/2+1}, w_3, w_{m/2+3}, w_5, w_{m/2+5}, \dots, w_{m/2-1}, w_{m-1})$ . If the  $m$  subwords are stored in two registers,  $P_m$  can be performed with a pair of MIX left and right instructions with a proper subword size. The MIX left generates the first half of  $P_m(S)$  and the MIX right generates the second half.

The general MIX operation  $MIX_{g,m,k}(S)$  is a permutation on a sequence  $S$  of  $n$  minimal subwords, where  $n = gmk$ .  $g$ ,  $m$ , and  $k$  are powers of 2, and  $m > 1$ . The general MIX operation first divides  $S$  into  $g$  subsequences of the same length:  $(S_0, S_1, \dots, S_{g-1})$ . Each subsequence consists of  $m$  subwords and each subword has  $k$  minimal subwords.  $P_m$  is then applied to each subsequence. The final permuted sequence  $MIX_{g,m,k}(S) = (P_m(S_0), P_m(S_1), \dots, P_m(S_{g-1}))$ .

The operation in Fig. 1 can be considered as a  $MIX_{1,8,1}$  operation on a sequence of eight minimal subwords. The original sequence stored in R1 and R2 can be represented as  $S = (w_0, w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (A, B, C, D, a, b, c, d)$ . All the eight minimal subwords are in one subsequence ( $g = 1$ ). The permuted sequence in R3 and R4 is  $MIX_{1,8,1}(S) = (P_8(S)) = (w_0, w_4, w_2, w_6, w_1, w_5, w_3, w_7) = (A, a, C, c, B, b, D, d)$ . The permutation  $P_8$  is performed with a pair of MIX instructions.

Fig. 2 illustrates three MIX operations on 16 minimal subwords, whose values range from 0 to 15. In the figure, each small box represents a minimal subword. The number at the top indicates the subword position in the sequence and the one at the bottom is the subword value, both in binary format. In

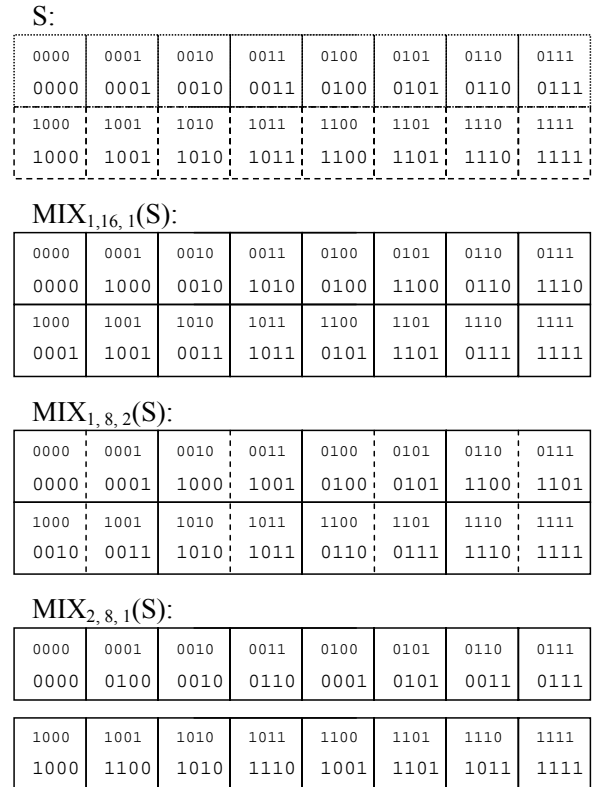


Fig. 2: Examples of MIX operations

the original sequence  $S$ , subwords are located in positions specified by their values. For example, subword 0000 is located at position 0000, subword 0001 is located at 0001, and so on. Doing so, we can tell the original position of a subword from its value.

The first operation in Fig. 2 is  $MIX_{1,16,1}(S)$ , which places all 16 minimal subwords in one single subsequence and permutes the subsequence with  $P_{16}$ . The result is  $P_{16}(S) = (w_0, w_8, w_2, w_{10}, w_4, w_{12}, w_6, w_{14}, w_1, w_9, w_3, w_{11}, w_5, w_{13}, w_7, w_{15}) = (0000, 1000, 0010, 1010, 0100, 1100, 0110, 1110, 0001, 1001, 0011, 1011, 0101, 1101, 0111, 1111)$ .

The second operation  $MIX_{1,8,2}(S)$  places the 16 minimal subwords in one single subsequence as well, but each subword now has two minimal subwords. The 8-subword subsequence is permuted with  $P_8$ .

The third operation  $MIX_{2,8,1}(S)$  divides  $S$  into two subsequences. The first subsequence consists of the first eight minimal subwords 0000 to 0111 and the second one consists of minimal subwords 1000 to 1111. Permutation  $P_8$  is then applied to each of them. The result of  $MIX_{2,8,1}(S)$  is the concatenation of two permuted sequences.

### B. Subword permutations with MIX operations

In MIX operations, a minimal subword goes to many different places, depending on the values of parameters  $g$ ,  $m$ , and  $k$ . The destination of a minimal subword can be obtained from its original position by exchanging a pair of bits in its index. Parameters  $g$ ,  $m$ , and  $k$  determine which pair of bits to be exchanged. In the examples shown in Fig. 2, the indices of the minimal subwords consist of four bits because there is a total of 16 minimal subwords. Let the four bits in the index be  $i_3, i_2, i_1$ , and  $i_0$ , and  $i_0$  be the least significant bit. The first operation  $MIX_{1,16,1}(S)$  exchanges  $i_3$  and  $i_0$ . The minimal subword 0001, for example, is moved from position 0001 ( $i_3 = 0$  and  $i_0 = 1$ ) to position 1000 ( $i_3 = 1$  and  $i_0 = 0$ ). Subword with  $i_0 = i_3$  stay in the same position. Such subwords include 0000, 0010, 0100, and 1111. Similarly  $MIX_{1,8,2}(S)$  exchanges  $i_3$  and  $i_1$ , and  $MIX_{2,8,1}(S)$  exchange  $i_2$  and  $i_0$ . Table I summarizes the parameter values of the MIX operations in Fig. 2 and the positions of the bits that are exchanged.

TABLE I: PARAMETERS OF MIX OPERATIONS AND EXCHANGED BIT PAIRS

$G$	$m$	$k$	Bit pair
$1 = 2^0$	$16 = 2^4$	$1 = 2^0$	3, 0
$1 = 2^0$	$8 = 2^3$	$2 = 2^1$	3, 1
$2 = 2^1$	$8 = 2^3$	$1 = 2^0$	2, 0

To understand why MIX operations switch two index bits, we first look at a simple MIX operation  $MIX_{1,m,1}$  on  $m$  minimal subwords where  $m$  is a power of 2.  $MIX_{1,m,1}$  is simply a permutation  $P_m$  on the  $m$  minimal subwords. There are  $\log_2 m$  bits in the minimal subword indices.  $P_m$  switches bit  $(\log_2 m - 1)$  and bit 0, as shown in Fig. 3a. The positions of the two exchanged bits are determined by the definition of  $P_m$ , which exchanges the highest and lowest bits of subword indices when  $m$  is a power of two.

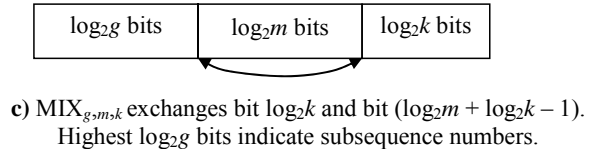
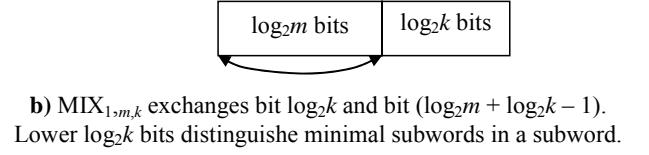
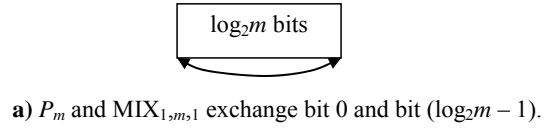


Fig. 3: Bits exchanged in a MIX operation

In  $MIX_{1,m,k}$  operations, there are  $mk$  minimal subwords in total. The minimal subword indices consist of  $\log_2 m + \log_2 k$  bits. The higher  $\log_2 m$  bits are subword numbers and the lower  $\log_2 k$  bits are minimal subword numbers within a subword. For  $k$  minimal subwords in the same subword, the higher  $\log_2 m$  index bits are the same but the lower  $\log_2 k$  bits are different. During  $MIX_{1,m,k}$  operations, the lower  $\log_2 k$  bits do not change because minimal subwords do not change relative positions in a subword. Among the higher  $\log_2 m$  bits, which are the subword numbers, the lowest and highest bits are exchanged. Essentially,  $MIX_{1,m,k}$  is still a  $P_m$  permutation, with a larger subword size. In minimal subword indices, the exchanged bits are bit  $\log_2 k$  and bit  $(\log_2 m + \log_2 k - 1)$ , as illustrated in Fig. 3b.

In  $MIX_{g,m,k}$  operations, the indices of minimal subwords can be divided into three sections. The highest  $\log_2 g$  bits are the subsequence numbers, the middle  $\log_2 m$  bits are the subword numbers, and the lowest  $\log_2 k$  bits are the minimal subword numbers. When  $MIX_{1,m,k}$  is performed on each of the  $g$  subsequences, the highest  $\log_2 g$  bits of the minimal subword indices remain the same. The exchanged bits are still bit  $\log_2 k$  and bit  $(\log_2 m + \log_2 k - 1)$ , which are the lowest and highest bits of the subword numbers. This is illustrated in Fig. 3c.

In general,  $MIX_{g,m,k}$  moves minimal subwords to a location where bits  $x$  and  $y$  of their indices are exchanged, and  $x = \log_2 m + \log_2 k - 1 = \log_2(mk) - 1$  and  $y = \log_2 k$ . In a sequence of  $n$  minimal subwords and  $n = 2^l$ , minimal subword indices consist of  $l$  bits,  $i_{l-1}, i_{l-2}, \dots, i_0$ . Choosing proper  $m$  and  $k$ , we can exchange any two bits in the index. Therefore, if a subword permutation can be obtained by permuting the index bits of minimal subwords, it can be performed with MIX operations. And at most  $\log_2 n - 1$  MIX operations are required to perform such subword permutations, where  $n$  is the number of minimal subwords, because the indices have  $\log_2 n$  bits and one MIX operation can move one index bit to its final location.

A MIX operation is the inverse of itself. If a MIX operation exchanges bit  $x$  and bit  $y$ , repeating this operation exchanges bit  $x$  and bit  $y$  again and moves minimal subwords to their original positions.

Taking matrix transpose as an example, we show how MIX operations can be chosen for a subword permutation. Suppose  $A$  is an  $8 \times 8$  matrix that need to be transposed.  $a_{ij}$  is the subword located at row  $i$  and column  $j$ . Both  $i$  and  $j$  are 3-bit numbers, whose values are between 0 and 7. We map  $A$  to a linear array  $S$  of 64 subwords. Subword  $a_{ij}$  becomes subword  $r$  in  $S$ , where  $r=8i+j=(i \ll 3)+j$ .  $r$  is actually the concatenation of the bits in  $i$  and  $j$ . The matrix transpose of  $A$  then becomes a subword permutation on  $S$ . Subword  $a_{ij}$  will be located at row  $j$  and column  $i$  in the transposed  $A$  and at position  $r'=(j \ll 3)+i$  in the permuted  $S$ . Comparing  $r$  and  $r'$ , we can see  $i$  and  $j$  are swapped. If the original location of a subword is  $(r_5r_4r_3r_2r_1r_0)$ , its new location is  $(r_2r_1r_0r_5r_4r_3)$ . Therefore, the  $8 \times 8$  matrix transpose can be achieved by a permutation on the bits in subwords' indices, which can be done with three MIX operations:

$$\begin{aligned} S &= \text{MIX}_{4,16,1}(S) && ; \text{exchange bit 0 and bit 3} \\ S &= \text{MIX}_{2,16,2}(S) && ; \text{exchange bit 1 and bit 4} \\ S &= \text{MIX}_{1,16,4}(S) && ; \text{exchange bit 2 and bit 5} \end{aligned}$$

The first MIX operation moves  $r_3$  to the new position and, at the same time, moves  $r_0$  to the right position as well. One MIX operation moves two bits. Similarly, other two MIX operations move two bits each. Three MIX operations move all the six bits and they can be performed in any orders. It does not matter which pair of bits are exchanged first. Although the  $8 \times 8$  matrix transpose needs only three MIX operations, in general, permutations of  $n$  subwords may require  $\log_2 n - 1$  MIX operations to place all index bits in proper positions. However, we will see in the next section that some MIX operations do not need to be performed explicitly.

#### IV. ISA SUPPORT FOR MIX OPERATIONS

There are numerous MIX operations as three parameters can be set to many different values. In this section, we investigate whether a small set of essential MIX instructions can perform all the MIX operations. We examine if the three parameters of MIX operations are *scalable*. A parameter is scalable if the ISA support of MIX operations with the parameter set to a specific value can also perform efficiently other MIX operations with the same parameter set to different values.

Parameter  $g$  is scalable when no two subsequences are stored in the same register. Since subsequences are in different registers, operations on a subsequence do not affect other subsequences. If  $\text{MIX}_{1,m,k}$  is supported,  $\text{MIX}_{g,m,k}$  can be done efficiently for any  $g \geq 1$ . We simply perform  $\text{MIX}_{1,m,k}$  on each of the  $g$  subsequences. If a register holds more than one subsequence, we have two options. We may add new MIX instructions for each MIX operation on one register, or we can use instructions for arbitrary subword or bit permutations.

The subword size  $k$  is not scalable. If a subword size is not supported by ISA, there is no general efficient way to process such subwords with instructions for other subword sizes. Note that when the subword size is equal to or greater than the word size, the MIX operation does not need to be performed because any subwords can already be accessed directly.

The subword number  $m$  is scalable when subsequences of  $m$

subwords are stored in two or more registers. Suppose each register stores  $R$  minimal subwords and the MIX left and MIX right instructions on minimal subwords are available. A pair of MIX left and MIX right instructions performs a  $\text{MIX}_{1,2R,1}$  operation. In this case, the  $m=2R$  subwords are stored in two registers.

When  $m$  increases, subwords have to be stored in more than two registers. Consider the  $\text{MIX}_{1,m,1}$  operation, where  $m = 2uR$  and  $u$  is a power of two. The  $m$  minimal subwords need  $2u$  registers because a register can hold only  $R$  minimal subwords. Suppose RS0 through RS( $2u-1$ ) are the source registers and RR0 through RR( $2u-1$ ) are the destination registers. In a  $\text{MIX}_{1,m,1}$  operation, the  $2u$  source registers are divided into two halves, as well as the  $2u$  destination registers. The four halves are illustrated in Fig. 4 as four rows, each of which consists of  $u$  registers. The top two rows are the source registers and the bottom two rows the destination registers. The registers also form  $u$  columns. The  $\text{MIX}_{1,m,1}$  operation takes the even-numbered minimal subwords, alternatively from two halves of the source registers, and place them into the first half of the result. Then, it takes the odd-numbered subwords and places them into the second half of the result. During this process, subwords do not cross columns. For example, the subwords in RS0 and RS( $u$ ) are placed into RR0 and RR( $u$ ), with even-numbered subwords in RR0 and odd-numbered subwords in RR( $u$ ). Within each column, the operation performed is actually a  $\text{MIX}_{1,2R,1}$ . Therefore,  $\text{MIX}_{1,m,1}$  can be performed with  $u$   $\text{MIX}_{1,2R,1}$  operations, one in each column. In total, we need  $2u$  instructions to perform  $\text{MIX}_{1,m,1}$  for  $m = 2uR$ .

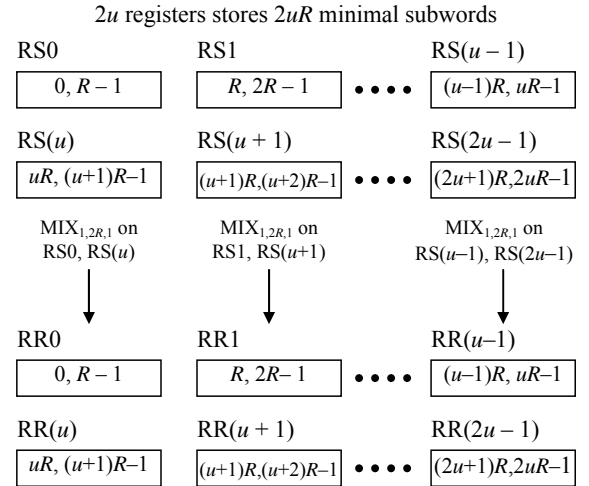


Fig. 4: Scalability of subword number  $m$

When the number of subwords  $m < 2R$ , all the subwords in a sequence is placed in one register. We have two options, which are similar to the solutions for storing more than one subsequence in one register (see the discussions of parameter  $g$ ). New instructions may be added for each different  $m$ . For example, IA-64 has MUX1@MIX for eight 8-bit subwords and MUX2@ALT for four 16-bit subwords. Alternatively, bit or subword permutation instructions can be used to perform the MIX operation within a register. For instance, PERMUTE

in PA-RISC 2.0 can perform arbitrary permutations, which include MIX operations, of four 16-bit subwords.

In summary,  $k$  is not scalable,  $g$  is scalable when no two subsequences share a register, and  $m$  is scalable when subsequences are stored in two or more registers. When  $m$  is scalable,  $g$  is scalable too. Since subword size  $k$  is not scalable, processors have to support in ISA all the needed subword sizes. When a subsequence is stored in more than one register, i.e., when both  $g$  and  $m$  are scalable, two variants of MIX instructions like the ones illustrated in Fig. 1 can perform  $MIX_{g,m,k}$  for any  $m > R$ , where  $R$  is the number of minimal subwords in a register and  $k$  is a supported subword size. When one or more complete subsequences are in a register, specific MIX instructions or general bit or subword permutation instructions are needed.

Table II summarizes the number of instructions for MIX operations. When the subword size  $k$  is equal to or greater than  $R$ , the MIX operations do not need to be performed explicitly because every subword can be accessed directly. When the subword size is smaller than  $R$ , there are two cases to consider. In the first case where a subsequence is stored in more than one register,  $MIX_{g,m,k}$  takes  $gmk/R$  instructions. Each subsequence, consisting of  $mk$  minimal subwords, is stored in  $mk/R$  registers, and needs  $mk/R$  instructions. In the second case, one or more subword subsequences are stored in one register. All the minimal subwords are stored in  $\lceil gmk/R \rceil$  registers, where  $\lceil x \rceil$  is the smallest integer not less than  $x$ . If  $I$  instructions are needed to perform MIX operations for the subwords in a register,  $MIX_{g,m,k}$  takes  $I \lceil gmk/R \rceil$  instructions.

The analysis in Section III showed that if a permutation of  $n$  minimal subwords can be achieved with MIX operations, then at most  $\log_2 n - 1$  MIX operations are needed. On processors that can store  $R$  minimal subwords in a register, the number of required MIX operations is reduced to  $\log_2 R$  because the MIX operations with  $k \geq R$  do not need to be performed explicitly.

TABLE II: NUMBER OF INSTRUCTIONS FOR MIX OPERATIONS

MIX parameters	Number of instructions	Description and positions of the exchanged bits, $x$ and $y$
$k \geq R^*$	0	The subword size is larger than the word size. $x \geq \log_2 R$ and $y \geq \log_2 R$ .
$k < R$ and $mk > R$	$gmk/R$	A subsequence is stored in two or more registers. $x \geq \log_2 R$ and $y < \log_2 R$ .
$mk \leq R$	$I \lceil gmk/R \rceil^+$	One or more subsequences are stored in one register. $x < \log_2 R$ and $y < \log_2 R$ .

\*  $R$  = Number of minimal subwords in a register.

+  $I$  = Number of instructions that perform MIX operations within a register.

## V. CONCLUSIONS

The MIX instruction is useful and effective in many applications to perform subword permutations. In this paper, we first defined MIX operations in a more general form. We then studied its properties and identified a class of subword permutations that can be performed with MIX operations. A subword permutation can be performed with MIX operations if the new locations of subwords can be obtained through a permutation on their original index bits. For this class of subword permutations, at most  $\log_2 n - 1$  MIX operations are

required for  $n$  minimal subwords. On processors that a register can hold  $R$  minimal subwords, the number of MIX operations that really need to be performed is at most  $\log_2 R$ . We also studied the ISA support for the generalized MIX operations. MIX instructions like those implemented in PA-RISC 2.0 and IA-64 can support MIX operations on sequences stored in two or more registers. In this case, a  $MIX_{g,m,k}$  operation takes  $gmk/R$  instructions. MIX operations on subwords stored in one register need either specific MIX instructions or general subword permutation instructions. If a register needs  $I$  instructions, a  $MIX_{g,m,k}$  operation takes  $I \lceil gmk/R \rceil$  instructions. The work in this paper provides programmers or compilers with a systematic method that performs subword permutations with MIX instructions. It also helps processor designers define MIX and subword permutation instructions, especially when designing application-specific ISAs.

## REFERENCES

- [1] R. B. Lee, "Subword parallelism in MAX-2," *IEEE Micro*, Vol. 16, No. 4, pp.51-59, August 1996.
- [2] R. B. Lee, "Multimedia extensions for general-purpose processors," *Proceedings of the IEEE Signal Processing Systems Design and Implementation*, pp. 9-23, November 1997.
- [3] A. Peleg and U. Weiser, "MMX technology extension to the Intel architecture," *IEEE Micro*, 16(4), pp. 10-20, August 1996.
- [4] K. Diefendorff, P. K. Dubey, R. Hochsprung, H. Scales, "Altivec extension to PowerPC accelerates media processing," *IEEE Micro*, Vol. 20, No. 2, pp. 85-95, April 2000.
- [5] Intel Corporation, *IA-64 Application Developers Architecture Guide*, Intel Corporation, May 1996.
- [6] R. B. Lee, *Efficient selection and mixing of multiple sub-word items packed into two or more computer words*, US Patent 5,673,321, filed June 29, 1995, granted September 30, 1997.
- [7] R. B. Lee, "Subword permutation instructions for two-dimensional multimedia processing in MicroSIMD architectures," *Proceedings of the IEEE International Conference on Application-specific Systems, Architectures and Processors*, pp. 3-14, July 2000.
- [8] Z. J. Shi and R. B. Lee, "Bit permutation instructions for accelerating software cryptography," *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 138-148, July, 2000.
- [9] X. Yang and R. B. Lee, "Fast subword permutation instructions using omega and flip network stages", *Proceedings of ICCD*, pp. 15-22, September 2000.
- [10] X. Yang, M. Vachharajani and R. B. Lee, "Fast subword permutation instructions based on butterfly networks", *Proceedings of Media Processors 1999 IS&T/SPIE Symposium on Electric Imaging: Science and Technology*, pp. 80-86, January 2000.
- [11] R. B. Lee, Z. J. Shi and X. Yang, "Efficient permutation instructions for fast software cryptography", *IEEE Micro*, Vol. 21, No. 6, pp. 56-69, December 2001.
- [12] Z. J. Shi, X. Yang and R. B. Lee, "Arbitrary bit permutations in one or two cycles", *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*, June 2003.
- [13] Z. J. Shi, *Bit permutation instructions: architecture, implementation, and cryptographic properties*, Ph.D. thesis, Princeton University, June 2004.
- [14] Z. Shi and R. B. Lee, "Subword sorting with versatile permutation instructions," *Proceedings of ICCD*, pp. 234-241, September 2002.