Vector (SIMD)Processing

Carl Hamacher, Zvonko Vranesic and Safwat Zaky, Computer Organization and Embedded Systems, (6e), McGraw Hill Publication, 2017.
Ch 12: 12.2

Vector (SIMD)Processing

- Many computationally demanding applications involve programs that use loops to perform operations on vectors of data, where a vector is an array of elements such as integers or floating-point numbers.
- When a processor executes the instructions in such a loop, the operations are performed one at a time on individual vector elements.
- Many instructions need to be executed to process all vector elements.
- A processor can be enhanced with multiple ALUs.
- It is possible to operate on multiple data elements in parallel using a single instruction.
- Such instructions are called single-instruction multiple-data (SIMD) instructions. They are also called vector instructions.
- These instructions can only be used when the operations performed in parallel are independent. This is known as data parallelism.
- The data for vector instructions are held in vector registers, each of which can hold several data elements. The number of elements, L, in each vector register is called the vector length.
- It determines the number of operations that can be performed in parallel on multiple ALUs.

Vector (SIMD)Processing

- The vector instruction

  VectorAdd.S Vi, Vj, Vk

- computes L sums using the elements in vector registers Vj and Vk, and places the resulting sums in vector register Vi.
- Suffix S denotes the size of each data element
- Special instructions are needed to transfer multiple data elements between a vector register and the memory. The instruction

  VectorLoad.S Vi, X(Rj)

- causes L consecutive elements beginning at memory location X + [Rj] to be loaded into vector register Vi. Similarly, the instruction

  VectorStore.S Vi, X(Rj)

- causes the contents of vector register Vi to be stored as L consecutive locations in the memory.

Vectorization

- In a source program written in a high-level language, loops that operate on arrays of integers or floating-point numbers are vectorizable if the operations performed in each pass are independent of the other passes.
- Using vector instructions reduces the number of instructions that need to be executed
- Enables the operations to be performed in parallel on multiple ALUs.
- A vectorizing compiler can recognize such loops, if they are not too complex, and generate vector instructions.

Vectorization Example

- Consider vectorization of the loop given below

```
for (i = 0; i < N; i++)
   A[i] = B[i] + C[i];
```

(a) A C-language loop to add vector elements

- Assume that the starting locations in memory for arrays A, B, and C are in registers R2, R3, and R4. Using conventional assembly-language instructions, the compiler may generate the loop.

```
        Move               R5, #N        R5 is the loop counter.
LOOP:   Load               R6, (R3)      R3 points to an element in array B.
        Load               R7, (R4)      R4 points to an element in array C.
        Add                R6, R6, R7    Add a pair of elements from the arrays.
        Store              R6, (R2)      R2 points to an element in array A.
        Add                R2, R2, #4    Increment the three array pointers.
        Add                R3, R3, #4
        Add                R4, R4, #4
        Subtract           R5, R5, #1    Decrement the loop counter.
        Branch_if_[R5]>0   LOOP          Repeat the loop if not finished.
```

(b) Assembly-language instructions for the loop

Vectorization Example Contd..

- The Load, Add, and Store instructions at the beginning of the loop are replaced by corresponding vector instructions that operate on L elements at a time.
- The vectorized loop requires only N/L passes to process all of the data in the arrays.
- With L elements processed in each pass through the loop, the address pointers in registers R2, R3, and R4 are incremented by 4L, and the count in register R5 is decremented by L.

Vectorization Example Contd..

- Vectorized form of the loop

	Move	R5, #N	R5 counts the number of elements to process.
LOOP:	VectorLoad.S	V0, (R3)	Load L elements from array B.
	VectorLoad.S	V1, (R4)	Load L elements from array C.
	VectorAdd.S	V0, V0, V1	Add L pairs of elements from the arrays.
	VectorStore.S	V0, (R2)	Store L elements to array A.
	Add	R2, R2, #4*L	Increment the array pointers by L words.
	Add	R3, R3, #4*L	
	Add	R4, R4, #4*L	
	Subtract	R5, R5, #L	Decrement the loop counter by L.
	Branch_if_[R5]> 0	LOOP	Repeat the loop if not finished.

(c) Vectorized form of the loop