Vector Processors

Slide courtesy: S. R. Sarangi

Slides adapted by: S. Mittal

Vector Processors

- A vector instruction operates on arrays of data
  - Example : There are vector instructions to add or multiply two arrays of data, and produce an array as output
  - **Advantage** : Can be used to perform all kinds of array, matrix, and linear algebra operations. These operations form the core of many scientific programs, high intensity graphics, and data analytics applications.

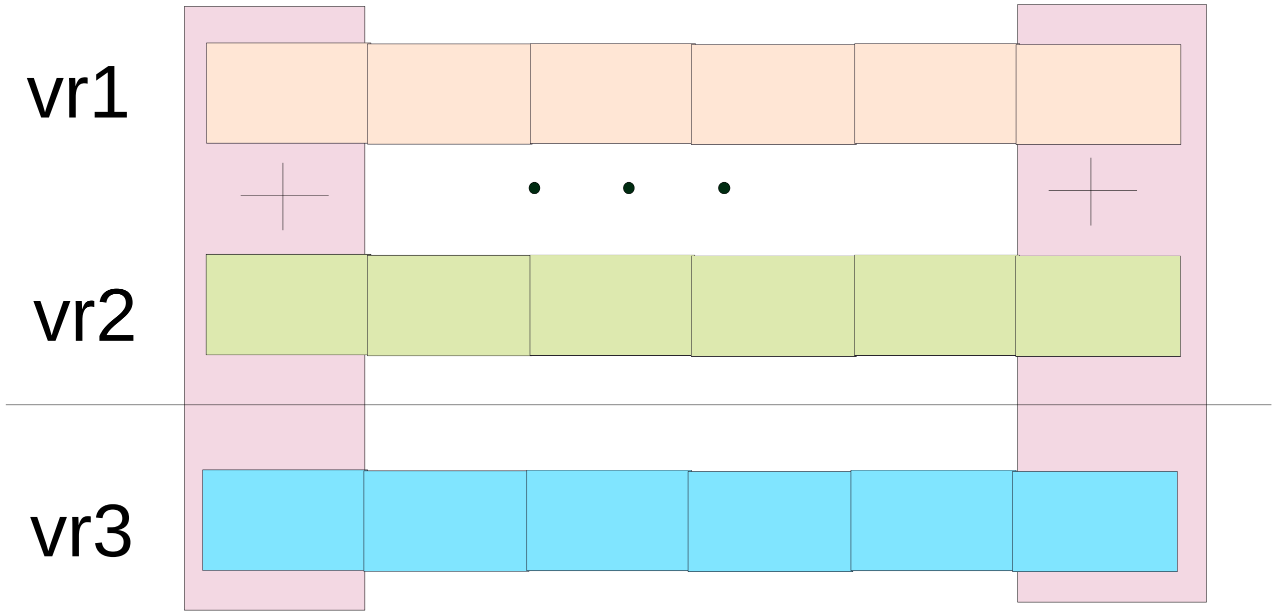
Background

- Vector processors were traditionally used in supercomputers
- Vector instructions gradually found their way into mainstream processors
  - MMX, SSE1, SSE2, SSE3, SSE4, and AVX instruction sets for x86 processors
  - AMD 3D Now Instruction Set

Software Interface

- Let us define a vector register
  - Example : 128 bit registers in the MMX instruction set → XMM0 … XMM15
  - Can hold 4 floating point values, or 8 2-byte short integers
  - Addition of vector registers is equivalent to pairwise addition of each of the individual elements.
  - The result is saved in a vector register of the same size.

Example of Vector Addition



Let us define 8 128 bit vector registers in SimpleRisc. vr0 ... vr7

Loading Vector Registers

- There are two **options** :
  - Option 1 : We assume that the data elements are stored in contiguous locations
  - Let us define the v.ld instruction that uses this assumption.

Instruction	Semantics
v.ld vr1, 12[r1]	vr1 ← ([r1+12], [r1+16],[r1+20], [r1+24])

  - Option 2: Assume that the elements are not saved in contiguous locations.

Scatter Gather Operation

- The data is scattered in memory
  - The load operation needs to gather the data and save it in a vector register.
- Let us define a scatter gather version of the load instruction → v.sg.ld
  - It uses another vector register that contains the addresses of each of the elements.

Instruction	Semantics
v.sg.ld vr1, vr2	vr1 ← ([vr2[0]], [vr2[1]], [vr2[2]], [vr2[3]])

Vector Store Operation

- We can similarly define two vector store operations

Instruction	Semantics
v.sg.st vr1, vr2	[vr2[0]] ← vr1[0] [vr2[1]] ← vr1[1] [vr2[2]] ← vr1[2] [vr2[3]] ← vr1[3]

Instruction	Semantics
v.st vr1, 12[r1]	[r1+12] ← vr1[0] [r1+16] ← vr1[1] [r1+20] ← vr1[2] [r1+24] ← vr1[3]

Vector Operations

- We can now define custom operations on vector registers
    - v.add → Adds two vector registers
    - v.mul → Multiplies two vector registers
    - We can even have operations that have a vector operand and a scalar operand → Multiply a vector with a scalar.

Predicated Instructions

- Suppose we want to run the following code snippet on each element of a vector register
  - if(x < 10) x = x + 10 ;
- Let the input vector register be vr1
- We first do a vector comparison :
  - v.cmp vr1, 10
  - It saves the results of the comparison in the **v.flags** register (vector form of the flags register)

Predicated Instructions - II

- If a condition is true, then the predicated instruction gets evaluated
  - Otherwise, it is replaced with a nop.
- Consider a scalar predicated instruction (in the ARM ISA)
  - addeq r1, r2, r3
  - r1 = r2 + r3 (if the previous comparison resulted in an equality)

Predicated Instructions - III

- Let us now define a vector form of the predicated instruction
  - For example : v.<p>.add (<p> is the predicate)
  - It is a regular add instruction for the elements in which the predicate is true.
  - For the rest of the elements, the instruction becomes a nop
- Example of predicates :
  - lt (less than) , gt (greater than), eq (equality)

Predicated Instructions - IV

* Implementation of our function :
  * if (x < 10) x = x + 20

```
v.cmp vr1, 10
v.lt.add vr1, vr1, 20
```

Adds 20 to every element of *vr1*

Design of a Vector Processor

- Salient Points
  - We have a vector register file and a scalar register file
  - There are scalar and vector functional units
  - Unless we are converting a vector to a scalar or vice versa, we in general do not forward values between vector and scalar instructions
  - The memory unit needs support for regular operations, vector operations, and possibly scatter-gather operations.