Computer Architecture

SIMD Architecture: Overview

Debadatta Mishra, CSE, IITK

Classification of Computing Frameworks

- Single Instruction Single Data (SISD)
  - Single Core with ILP techniques such as superscalar and speculative execution

Classification of Computing Frameworks

- Single Instruction Single Data (SISD)
  - Single Core with ILP techniques such as superscalar and speculative execution
- Single Instruction Multiple Data (SIMD)
  - Multiple SIMD processor execute the same instruction on multiple data to exploit data level parallelism

Classification of Computing Frameworks

- Single Instruction Single Data (SISD)
  - Single Core with ILP techniques such as superscalar and speculative execution
- Single Instruction Multiple Data (SIMD)
  - Multiple SIMD processor execute the same instruction on multiple data to exploit data level parallelism
- Multiple Instruction Single Data (MISD)
  - Not very common as concurrent operation on the same data ensuring correctness is difficult to achieve

Classification of Computing Frameworks

- Single Instruction Single Data (SISD)
  - Single Core with ILP techniques such as superscalar and speculative execution
- Single Instruction Multiple Data (SIMD)
  - Multiple SIMD processor execute the same instruction on multiple data to exploit data level parallelism
- Multiple Instruction Single Data (MISD)
  - Not very common as concurrent operation on the same data ensuring correctness is difficult to achieve
- Multiple Instruction Multiple Data (MIMD)
  - Independent processing by different processors with correctness guarantees
  - Multi-core and multi-threaded processors

Classification of Computing Frameworks

- Single Instruction Single Data (SISD)
  - Single Core with ILP techniques such as superscalar and speculative execution

Covered

- Single Instruction Multiple Data (SIMD)
  - Multiple SIMD processor execute the same instruction on multiple data to e
    data level parallelism

Agenda for remaining lectures

- Multiple Instruction Single Data (MISD)
  - Not very common as concurrent operation on the same data ensuring correctness is difficult to achieve

- Multiple Instruction Multiple Data (MIMD)
  - Independent processing by different processors with correctness guarantees
  - Multi-core and multi-threaded processors

CS423

Overview of SIMD processing

- Same operation on multiple data items. Example: ***Y*** = *a* * ***X*** + ***Y*** ***X*** and ***Y*** are vectors and *a* is a scalar. Implementation using SISD?

Overview of SIMD processing

- Same operation on multiple data items. Example: ***Y*** = *a* * ***X*** + ***Y*** ***X*** and ***Y*** are vectors and *a* is a scalar.

SISD

```
// r1 = &x[0], r2 = &y[0], element size = 8
// size of the vectors = 64

ldD f0,(a)           //load scalar a
daddiu r3,r1, #512   //r3=&X[0] + 512
loop: ldD f1,0(r1)   // f1 = x[i]
mulD f1,f1,f0        // f1 = x[i]*a
ldD f2,0(r2)         // f2 = y[i]
addD f2,f1,f2        // f2 = y[i]+a*x[i]
sD f2,0(r2)          // y[i] = y[i]+a*x[i]
daddiu r1,r1,#8      // r1 = &x[i+1]
daddiu r2,r2,#8      // r2 = &y[i+1]
sub r4,r3,r1         // elements remaining
benz r4, loop
```

- #of instructions executed?
- Branch related instructions?
- Dependencies?

Overview of SIMD processing

- Same operation on multiple data items. Example: ***Y*** = *a* * ***X*** + ***Y*** ***X*** and ***Y*** are vectors and *a* is a scalar.

SISD

```
// r1 = &x[0], r2 = &y[0], element size = 8
// size of the vectors = 64

ldD f0,(a)           //load scalar a
daddiu r3,r1, #512   //r3=&X[0] + 512
loop: ldD f1,0(r1)   // f1 = x[i]
mulD f1,f1,f0        // f1 = x[i]*a
ldD f2,0(r2)         // f2 = y[i]
addD f2,f1,f2        // f2 = y[i]+a*x[i]
sD f2,0(r2)          // y[i] = y[i]+a*x[i]
daddiu r1,r1,#8      // r1 = &x[i+1]
daddiu r2,r2,#8      // r2 = &y[i+1]
sub r4,r3,r1         // elements remaining
benz r4, loop
```

- #of instructions executed? 578
- Branch related instructions? 4 out of 9 in each iteration of the loop
- Dependencies? Multiple dependencies in each iteration

Overview of SIMD processing

- Same operation on multiple data items. Example: ***Y*** = *a* * ***X*** + ***Y*** ***X*** and ***Y*** are vectors and *a* is a scalar.

SISD

```
// r1 = &x[0], r2 = &y[0], element size = 8
// size of the vectors = 64

ldD f0,(a)          //load scalar a
daddiu r3,r1, #512  //r3=&X[0] + 512
loop: ldD f1,0(r1)  // f1 = x[i]
mulD f1,f1,f0       // f1 = x[i]*a
ldD f2,0(r2)        // f2 = y[i]
addD f2,f1,f2       // f2 = y[i]+a*x[i]
sD f2,0(r2)         // y[i] = y[i]+a*x[i]
daddiu r1,r1,#8     // r1 = &x[i+1]
daddiu r2,r2,#8     // r2 = &y[i+1]
sub r4,r3,r1        // elements remaining
benz r4, loop
```

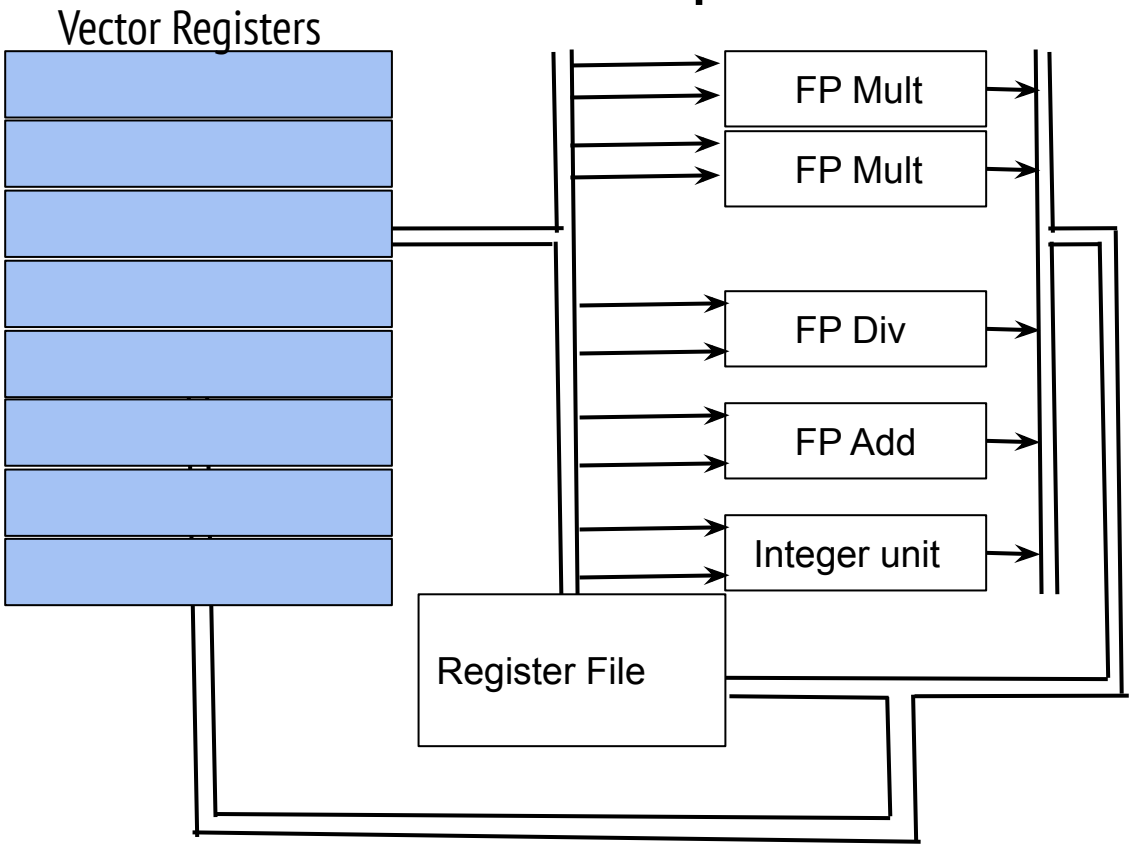
SIMD (using vector processing)

```
// r1 = &x[0], r2 = &y[0], element size = 8
// size of the vectors = 64

ldD f0,(a)          //load scalar a
ldV v1,r1           //load X to vector reg v1
mulVS v2,v1,f0      //v2 = a*X
ldV v3,r2           //load Y to vector reg v3
addVV v4,v3,v2      // v4 = a*X + Y
SV V4,r2            // Y = a * X + Y
```

- Reduction in #of instructions
- Data forwarding through chaining (more about thing in coming slides)

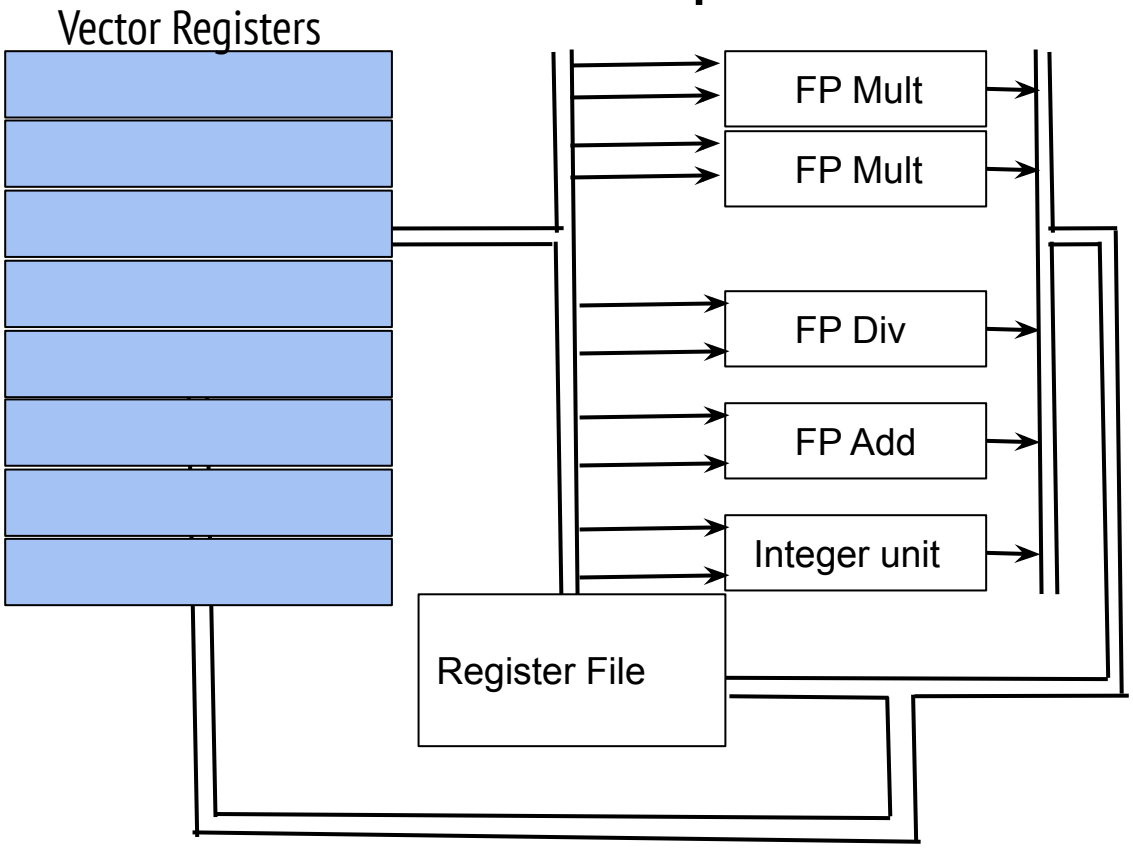
Structure of a vector processor



Vector registers

- Each register holds a single vector of length N
- Each element of the vector can be a 64-bit value
- Considering 8 vector registers of length 64, how many read/write ports are required?

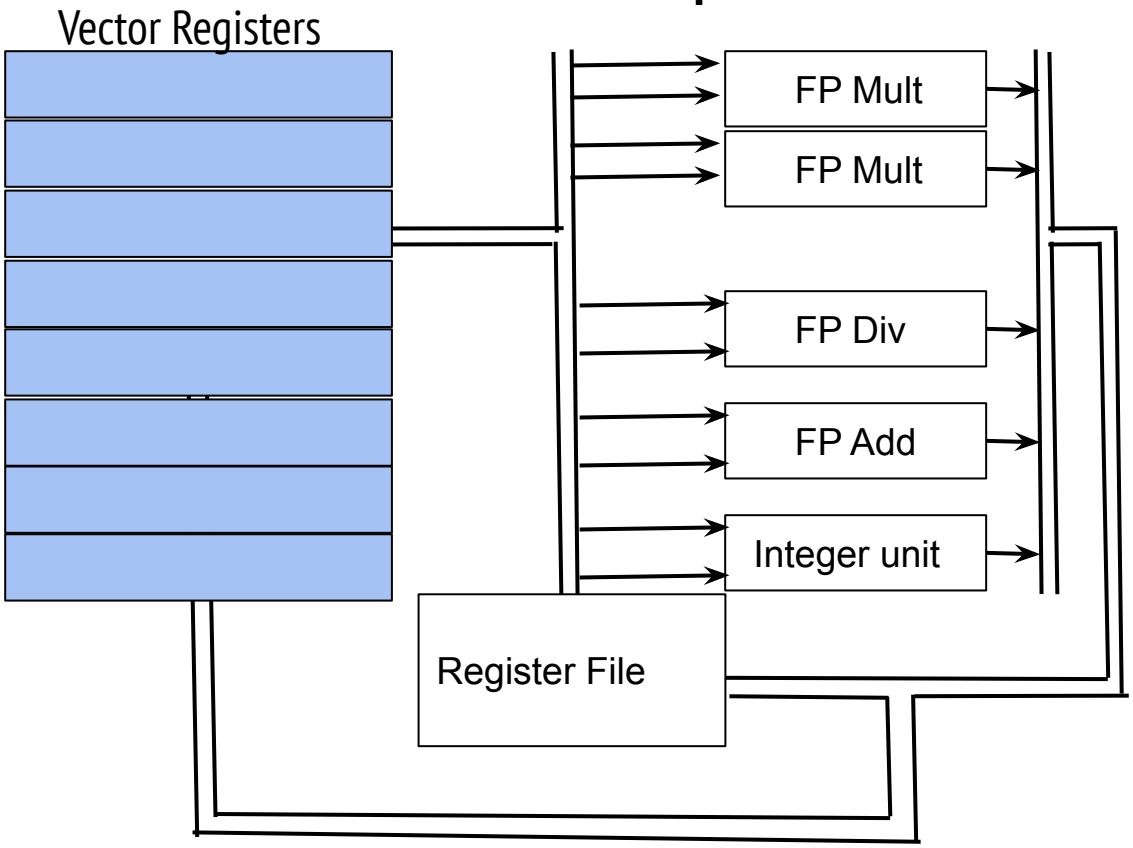
Structure of a vector processor



Special vector registers

- Why special vector registers are needed?

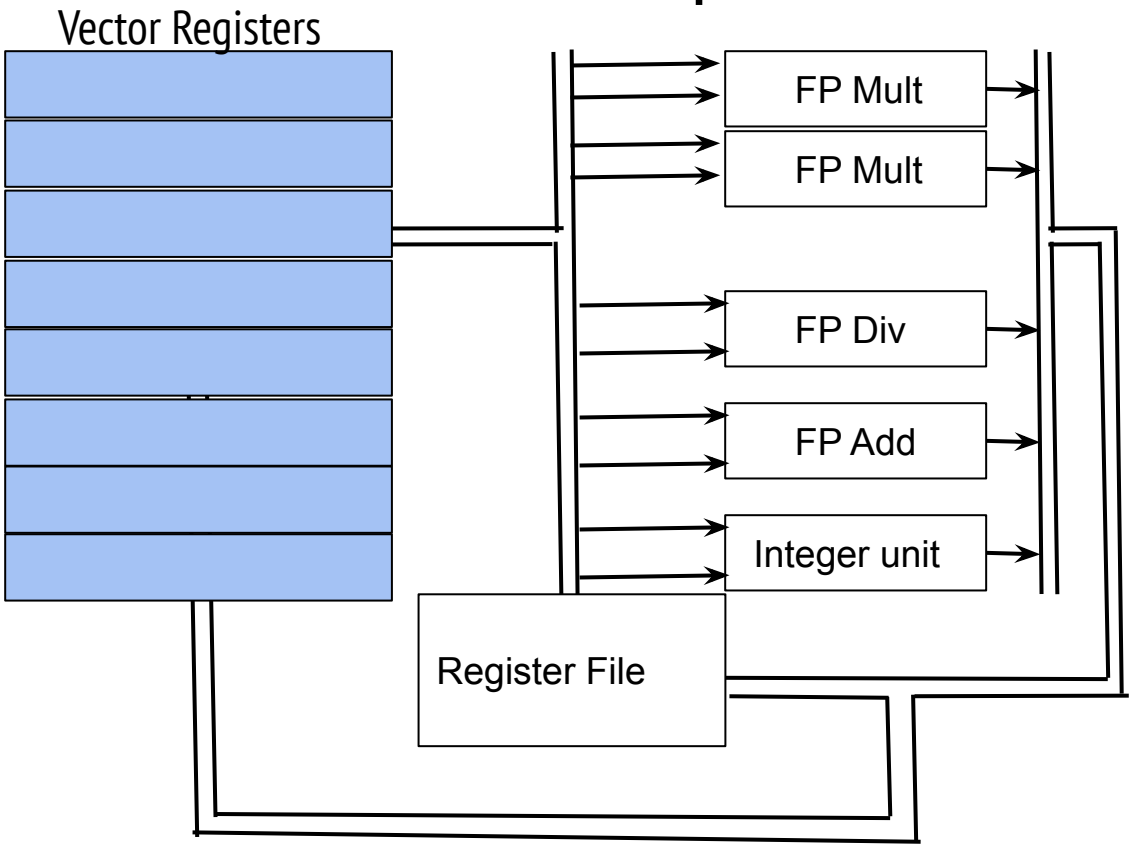
Structure of a vector processor



Special vector registers

- Why special vector registers are needed?
- VLEN: Vector length register to handle variable length vector operations
- VMASK: Mask register to support conditional execution. Typically a boolean vector of size N

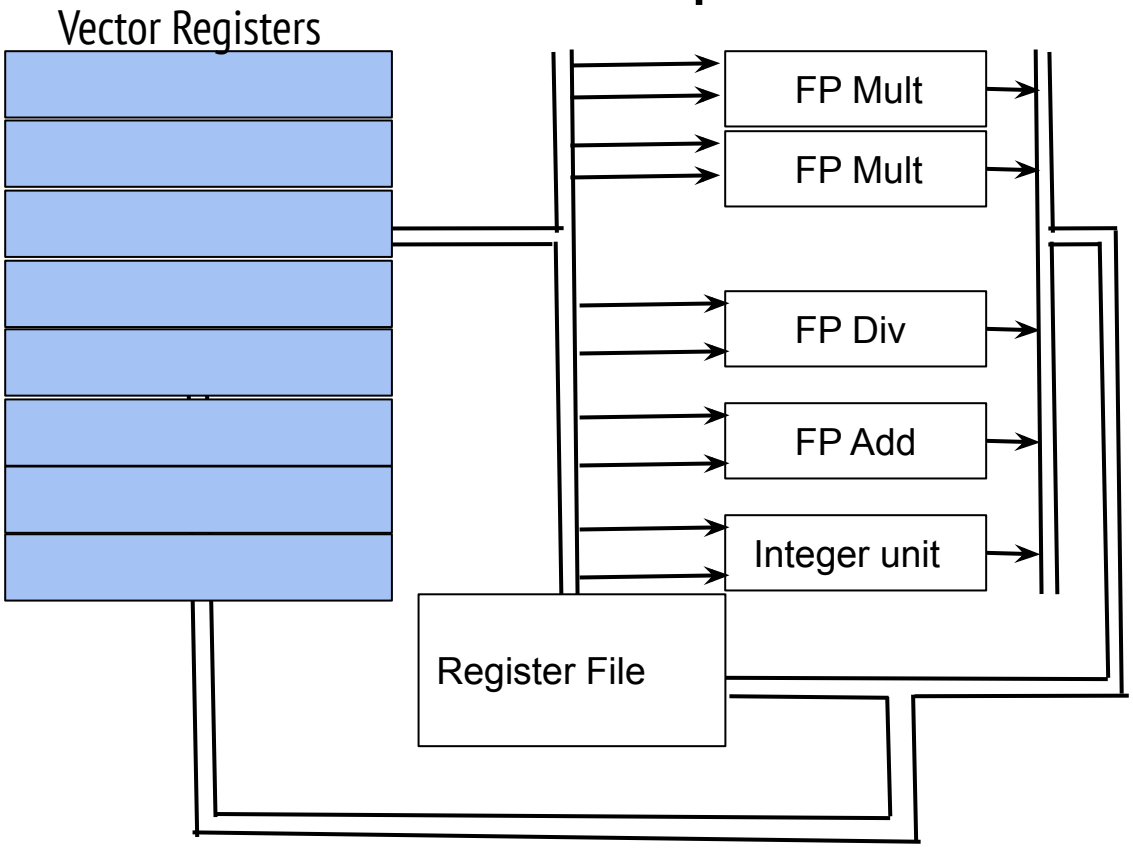
Structure of a vector processor



Vector functional units

- Each FU is pipelined with control units (for hazard detection)
- After initial fill, the can complete one operation every cycle
- Can be single lane or multiple lanes of pipeline

Structure of a vector processor



Vector load/store unit

- Pipelined load/store unit
- After the initial latency, one element can be moved between VR and memory in every cycle

Overview of Vector processing

- Same operation on multiple data items. Example: ***Y*** = *a* * ***X*** + ***Y*** ***X*** and ***Y*** are vectors and *a* is a scalar.

SIMD (using vector processing)

```
// r1 = &x[0], r2 = &y[0], element size = 8
// size of the vectors = 64

ldD f0,(a)          //load scalar a
ldV v1,r1           //load X to vector reg v1
mulVS v2,v1,f0      //v2 = a*X
ldV v3,r2           //load Y to vector reg v3
addVV v4,v3,v2      // v4 = a*X + Y
SV V4,r2            // Y = a * X + Y
```

- How a single vector instruction is executed?
- Assuming a 5-stage multiplier, how many cycles to execute “mulVS” instruction for a vector of size 64?

Overview of Vector processing

- Same operation on multiple data items. Example: ***Y*** = *a* * ***X*** + ***Y*** ***X*** and ***Y*** are vectors and *a* is a scalar.

SIMD (using vector processing)

```
// r1 = &x[0], r2 = &y[0], element size = 8
// size of the vectors = 64

ldD f0,(a)          //load scalar a
ldV v1,r1           //load X to vector reg v1
mulVS v2,v1,f0      //v2 = a*X
ldV v3,r2           //load Y to vector reg v3
addVV v4,v3,v2      // v4 = a*X + Y
SV V4,r2            // Y = a * X + Y
```

- Which instructions can be issued and executed in a concurrent manner?

Overview of Vector processing

- Same operation on multiple data items. Example: ***Y*** = *a* * ***X*** + ***Y*** ***X*** and ***Y*** are vectors and *a* is a scalar.

SIMD (using vector processing)

```
// r1 = &x[0], r2 = &y[0], element size = 8
// size of the vectors = 64

ldD f0,(a)          //load scalar a
ldV v1,r1           //load X to vector reg v1
mulVS v2,v1,f0      //v2 = a*X
ldV v3,r2           //load Y to vector reg v3
addVV v4,v3,v2      // v4 = a*X + Y
SV V4,r2            // Y = a * X + Y
```

- Which instructions can be issued and executed in a concurrent manner?
- Instructions not causing any structural hazard can be grouped to convoys
- How many convoys?

Overview of Vector processing

- Same operation on multiple data items. Example: ***Y*** = *a* * ***X*** + ***Y*** ***X*** and ***Y*** are vectors and *a* is a scalar.

SIMD (using vector processing)

```
// r1 = &x[0], r2 = &y[0], element size = 8
// size of the vectors = 64

ldD f0,(a)          //load scalar a
ldV v1,r1           //load X to vector reg v1
mulVS v2,v1,f0      //v2 = a*X
ldV v3,r2           //load Y to vector reg v3
addVV v4,v3,v2      // v4 = a*X + Y
SV V4,r2            // Y = a * X + Y
```

- Which instructions can be issued and executed in a concurrent manner?
- Instructions not causing any structural hazard can be grouped to convoys
- How many convoys? 3

Overview of Vector processing

- Same operation on multiple data items. Example: ***Y*** = *a* * ***X*** + ***Y*** ***X*** and ***Y*** are vectors and *a* is a scalar.

SIMD (using vector processing)

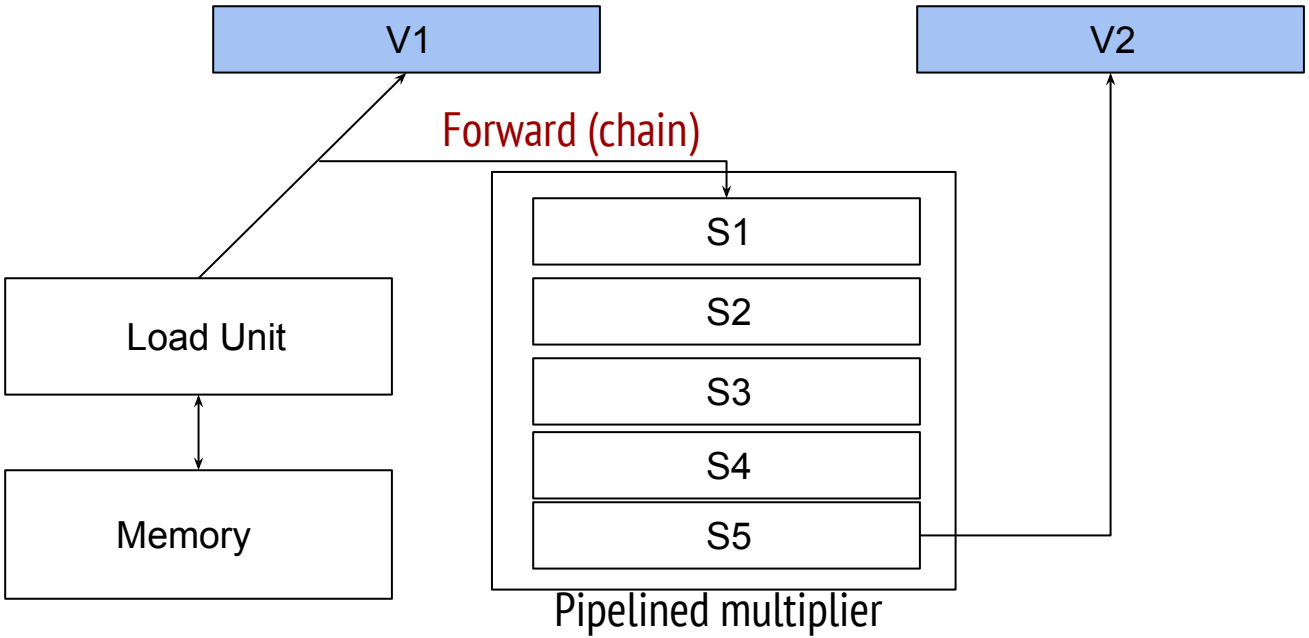
```
// r1 = &x[0], r2 = &y[0], element size = 8
// size of the vectors = 64

ldD f0,(a)          //load scalar a
ldV v1,r1           //load X to vector reg v1
mulVS v2,v1,f0      //v2 = a*X
ldV v3,r2           //load Y to vector reg v3
addVV v4,v3,v2      // v4 = a*X + Y
SV V4,r2            // Y = a * X + Y
```

- Which instructions can be issued and executed in a concurrent manner?
- Instructions not causing any structural hazard can be grouped to convoys
- How many convoys? 3
- How data forwarding works between dependent instructions?

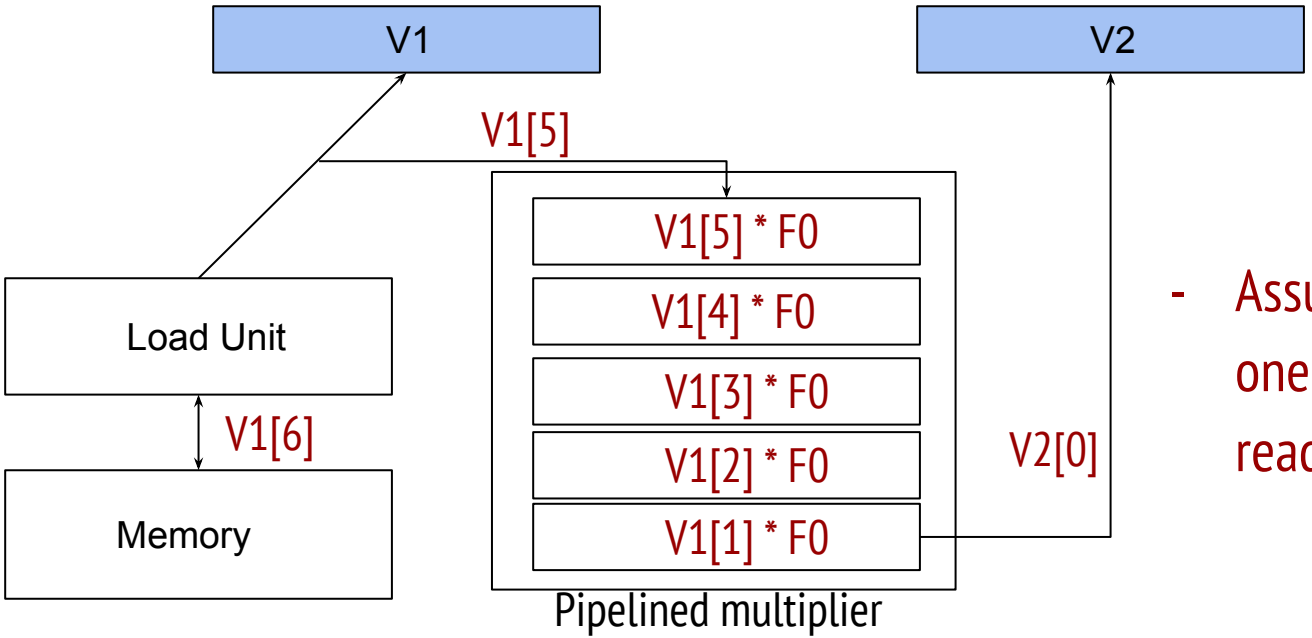
Vector processing

```
ldV v1,r1          //load X to vector reg v1
mulVS v2,v1,f0     //v2 = a*X
```



Vector processing

```
ldV v1,r1          //load X to vector reg v1
mulVS v2,v1,f0     //v2 = a*X
```



- Assuming single lane, one element in V2 is ready in each cycle

Overview of Vector processing

- Same operation on multiple data items. Example: ***Y*** = *a* * ***X*** + ***Y*** ***X*** and ***Y*** are vectors and *a* is a scalar.

SIMD (using vector processing)

```
// r1 = &x[0], r2 = &y[0], element size = 8
// size of the vectors = 64

ldD f0,(a)          //load scalar a
ldV v1,r1           //load X to vector reg v1
mulVS v2,v1,f0      //v2 = a*X
ldV v3,r2           //load Y to vector reg v3
addVV v4,v3,v2      // v4 = a*X + Y
SV V4,r2            // Y = a * X + Y
```

- Which instructions can be issued and executed in a concurrent manner?
- Instructions not causing any structural hazard can be grouped to convoys
- How many convoys? 3
- How data forwarding works between dependent instructions? Chaining techniques used to forward data