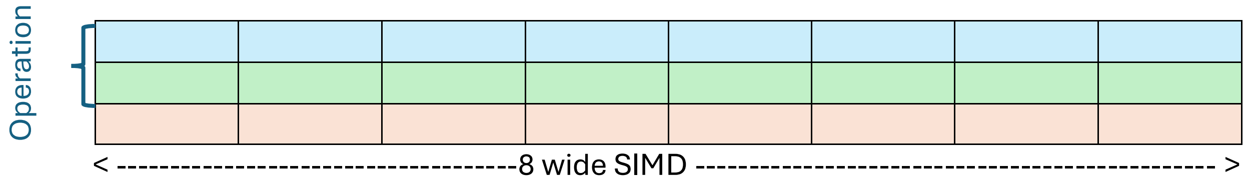
Description of the APIs:

The illustration below shows an 8 wide-SIMD engine, that computes on src1, src2 (8 elements at a time) and stores the result in dest. (where dest = src1 operator scr2). *This figure is only for illustration, the SIMD width depends on the target PIM architecture.*


Operation
< ------------------------------------8 wide SIMD -- >

8 elements of src1

8 elements of src2 <optional/scalar>

8 elements of dest

You will use one of the following APIs in this assignment.

Logic and Arithmetic Operation (Element-wise)

PimStatus **pimAdd**(PimObjId src1, PimObjId src2, PimObjId dest);

PimStatus **pimSub**(PimObjId src1, PimObjId src2, PimObjId dest);

PimStatus **pimMul**(PimObjId src1, PimObjId src2, PimObjId dest);

PimStatus **pimDiv**(PimObjId src1, PimObjId src2, PimObjId dest);

PimStatus **pimAbs**(PimObjId src, PimObjId dest);

Explanation: `dest = absolute(src)`

PimStatus **pimNot**(PimObjId src, PimObjId dest);

Explanation: `dest = not(src)`

PimStatus **pimAnd**(PimObjId src1, PimObjId src2, PimObjId dest);

PimStatus **pimOr**(PimObjId src1, PimObjId src2, PimObjId dest);

PimStatus **pimXor**(PimObjId src1, PimObjId src2, PimObjId dest);

PimStatus **pimXnor**(PimObjId src1, PimObjId src2, PimObjId dest);

PimStatus **pimGT**(PimObjId src1, PimObjId src2, PimObjId dest);

Explanation:

```
for i in range len(src1):
     dest[i] = src1[i] > src2[i] ? 1 : 0
```

PimStatus **pimLT**(PimObjId src1, PimObjId src2, PimObjId dest);

Explanation:

```
for i in range len(src1):
     dest[i] = src1[i] < src2[i] ? 1: 0
```

PimStatus **pimEQ**(PimObjId src1, PimObjId src2, PimObjId dest);

Explanation:

```
for i in range len(src1):
     dest[i] = src1[i] == src2[i] ? 1 : 0
```

PimStatus **pimNE**(PimObjId src1, PimObjId src2, PimObjId dest);

Explanation:

```
for i in range len(src1):
     dest[i] = src1[i] != src2[i] ? 1 : 0
```

PimStatus **pimMin**(PimObjId src1, PimObjId src2, PimObjId dest);

Explanation:

```
for i in range len(src1):
     dest[i] = min(src1[i], src2[i])
```

PimStatus **pimMax**(PimObjId src1, PimObjId src2, PimObjId dest);

Explanation:

```
for i in range len(src1):
     dest[i] = max(src1[i], src2[i])
```

Operations with scalar, all the elements of *src* are operated on using the *scalar values*

```
for i in range len(src):
          dest[i] = src1[i] <operation> scalarValue
```

PimStatus **pimAddScalar**(PimObjId src, PimObjId dest, uint64_t scalarValue);

PimStatus **pimSubScalar(**PimObjId src, PimObjId dest, uint64_t scalarValue);

PimStatus **pimMulScalar**(PimObjId src, PimObjId dest, uint64_t scalarValue);

PimStatus **pimDivScalar**(PimObjId src, PimObjId dest, uint64_t scalarValue);

multiply src1 with scalarValue and add the multiplication result with src2. Save the result to dest

PimStatus **pimScaledAdd**(PimObjId src1, PimObjId src2, PimObjId dest, uint64_t scalarValue);

Explanation:
```
for i in range len(src1):
          dest[i] = scalarValue * src1[i] + src2[i]
```

Reduction APIs

Note: Reduction sum range is [idxBegin, idxEnd)

PimStatus **pimRedSum**(PimObjId src, void* sum,uint64_t idxBegin = 0,uint64_t idxEnd = 0);

Explanation: add all the elements of the `src` and store the result in `sum`

PimStatus **pimRedMin**(PimObjId src, void* min, uint64_t idxBegin = 0, uint64_t idxEnd = 0);

Explanation: finds the minimum of the all the elements from `src` and store in `min`

PimStatus **pimRedMax**(PimObjId src, void* max,uint64_t idxBegin = 0, uint64_t idxEnd = 0);

Explanation: finds the maximum of the all the elements from `src` and store in `max`

More APIs can be found here: https://github.com/UVA-LavaLab/PIMeval-PIMbench/blob/main/libpimeval/src/libpimeval.h