Domain Specific Hardware Accelerators.

Vector Negate and Statistic Minima

CS6230: Initial Project Plan

CS19M014 Ashwini Tagadghar
CS20M001 Shailesh Tiwary
ME17B174 Sooryakiran

Overview.

The main objective of our project is to design vector accelerators for elementwise negation and statistics minima. These accelerators are connected to a bus and have Direct Memory Access, further lessening the workload on the CPU. The CPU issues instructions to the accelerators, which comprises pointers to data in the memory. There is no direct data transfer between the CPU and the accelerators. The result of the vector computation is written back to the memory. The Control Status Registers in these accelerators let the CPU control and read status from them.

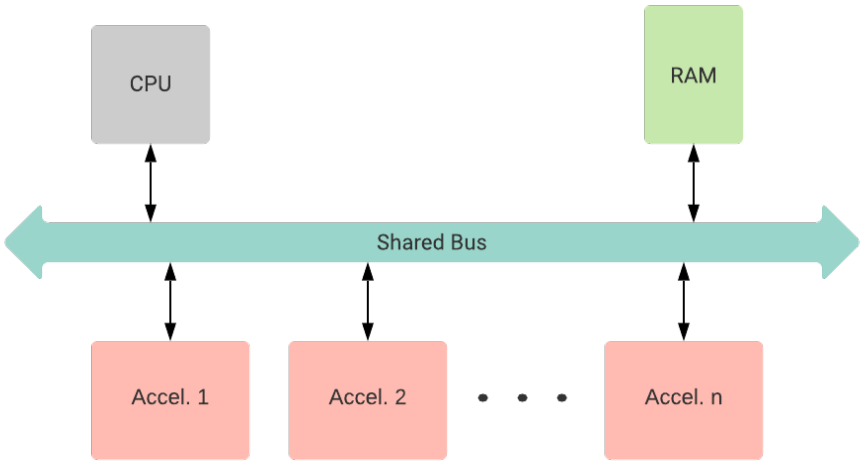


Figure 1. Overview of entire setup

Each accelerator corresponds to a specific vector function. In our case, one of the accelerators will accommodate the hardware to compute the statistics minima concerning float32, int8, int16 & int32, and the other accelerator will include the hardware for vector negation.

We will also make a minimal in order 2 stage pipelined CPU to issue instructions and a memory module.

General Structure of an Accelerator.

The proposed general architecture for a vector accelerator is as given in the figure. It is to be remarked that slight alterations may be required according to the purpose. While such a formation is suitable for most vector operations, matrix operations may require a different kind of design with systolic arrays. Since we intend to devise a design for accelerators concerning vector negation and statistics minimum, we restrict the general design for vector operations.

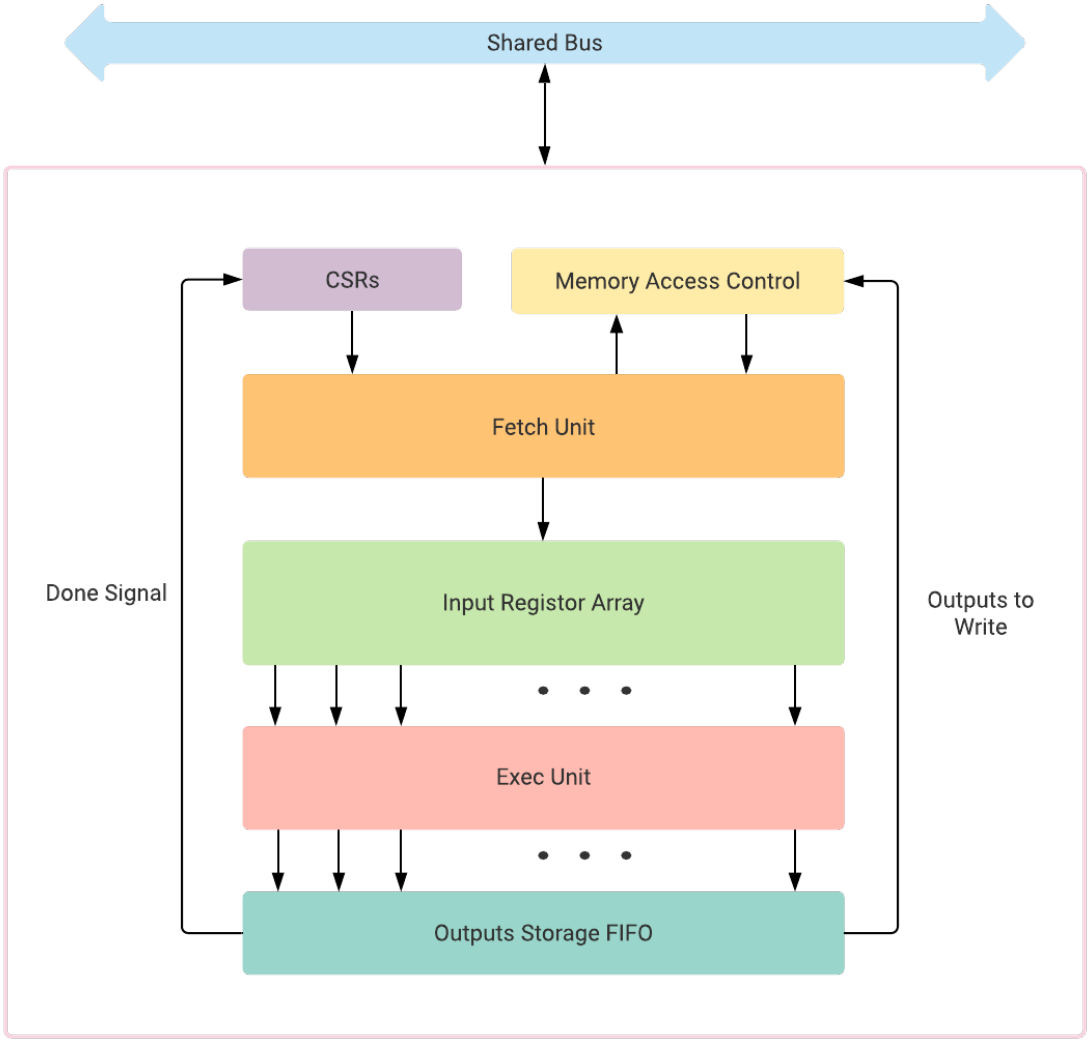


Figure 2. General architecture of an accelerator

The CPU writes the pointers for the source & destination vectors and the vector size into the accelerators' CSRs. Once the CPU instructs the accelerator to start computation by writing to the start flag of the CSRs, the fetch unit requests the memory access controller for the data. The memory access controller manages read/write requests and responses between the accelerator and the bus. The memory access controller prioritizes write-requests to memory to prevent a possible deadlock and stalling of the pipeline. Once the fetch unit gets the data, it gets pushed onto the temporary storage arrays.

We propose to implement temporary storage using 8-bit FIFOs. According to the instruction, the outputs can be interpreted according to the data type, as shown in the figure, and can be gathered by the appropriate execution unit. The width of the complete unit depends on the highest bandwidth supported through the bus. The representation of an 8-bit vector in the temporary storage unit is shown in the figure.

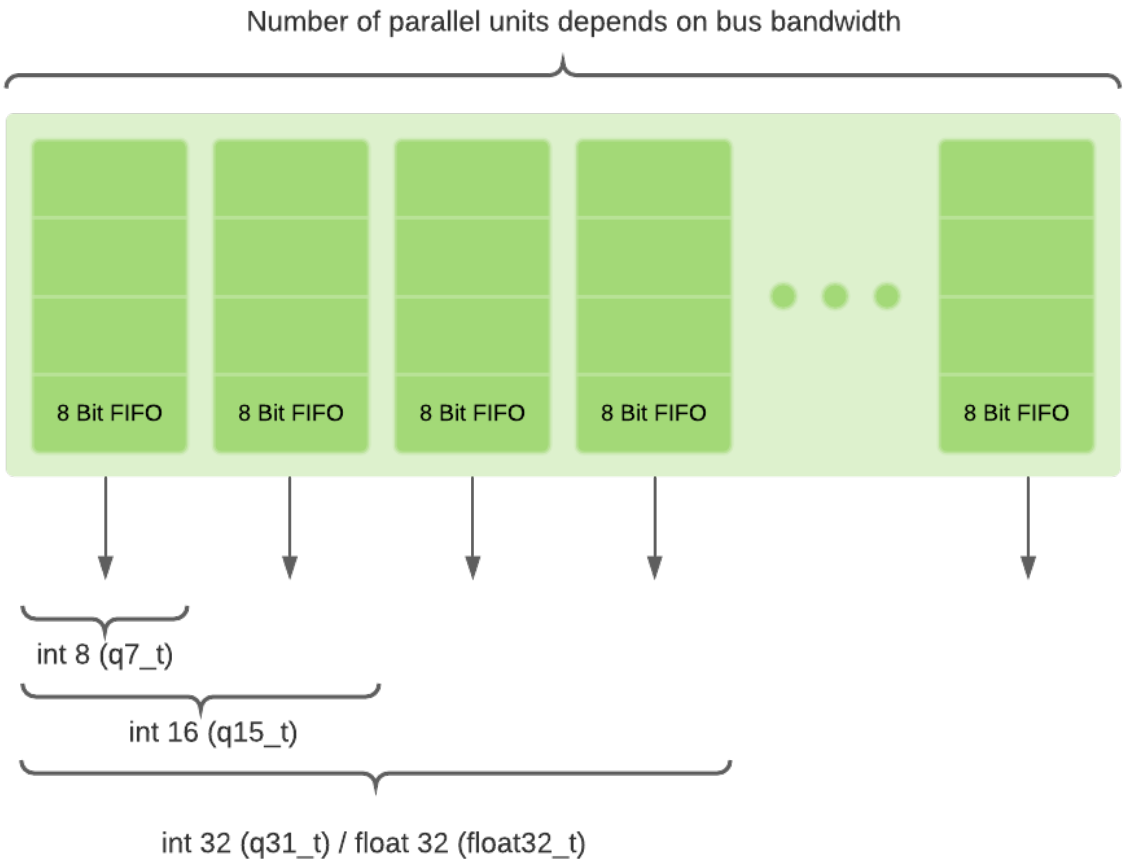


Figure 3a. Temporary storage unit, structural representation

Sample representation of an int 8 (q7_t) vector in a 64 bit wide storage unit.

```
q7_t A[14];
```

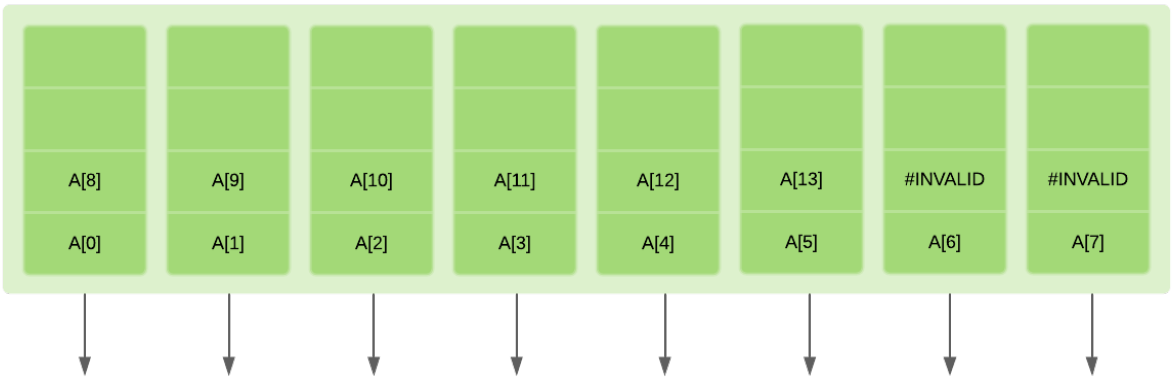


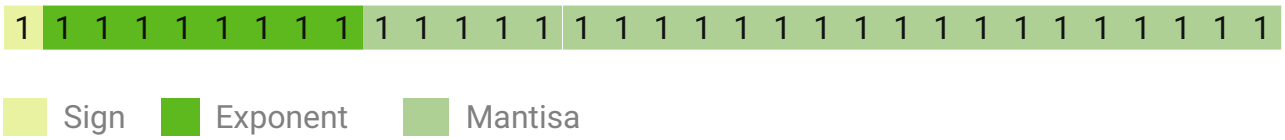
Figure 3b. Temporary storage unit, functional representation

Vector Negation Execution Unit.

The vector negation function implementation takes a vector of float32, int8, int16, or int32 numbers as input and produces a vector of the same size as output with elementwise negation. In floating-point numbers, the output has flipped sign bits of each member in the original vector.

This function would provide a routine for negating a floating-point vector. The interpretation of a floating-point number has been derived from the IEEE 754 standard. The number representation is as depicted below. The most significant bit (MSB) is towards the left.

The number representation is as depicted below. The most significant bit (MSB) is towards the left



The sign of the floating-point number is represented using a single bit. A 1 bit indicates a negative number, and a 0 indicates a positive number.

The floating-point number representation in binary can be generalized as

$$(-1)S \times M \times 2^{E}$$

here S is sign, M is mantissa or significand and E is exponent.

The aforesaid routine processes n elements from the vector in parallel, where n depends on the width of the bus data. Also, BlueSpec (via int8, int16 and int32) inherently supports negation of signed integers using the 2's complement representation, and thus no explicit steps are required for the same.

Vector Statistics Minima Execution Unit.

Here, we need to find out the minimum element present in the provided vector. The number of inputs that can be compared concurrently depends upon the bus bandwidth and input data type. For instance, if the bus bandwidth is 64-bit, we have 64-bit wide storage, and if the input data type is q7_t:

The number of elemens that can be executed simultaneosly are:
Bus bandwidth / sizeof(data type) = 64 / 8 = 8.

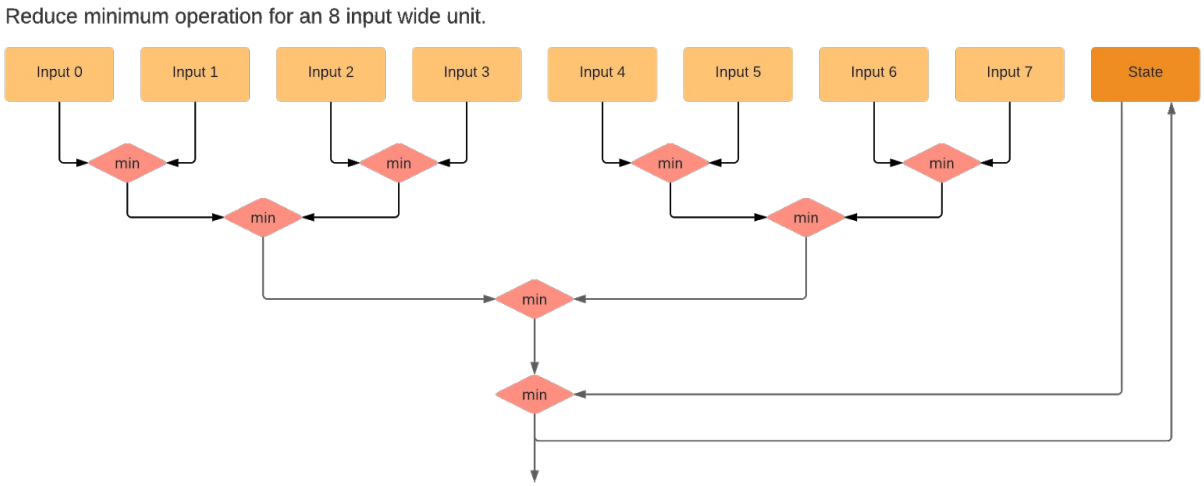


Figure 4. Exec. unit for a statistic minimum operation unit.

We take the first set of inputs and compare two consecutive inputs, to find the minimum without any overlap between inputs using the min function in Bluespec.

Min: Returns the minimum of the values x and y.
`function data_t min (data_t x, data_t y);`

State: Is a register initialized, resets to `INF` for each new vector. `INF` value is hardcoded with the highest possible value.

We find the overall minimum value from the first set of input is compared with the state value, updates the state value with a minimum if the minimum is less than the state value. This process is repeated for the remainder of the inputs. The final state value equals the statistics minima.

Auxilary Modules: CPU.

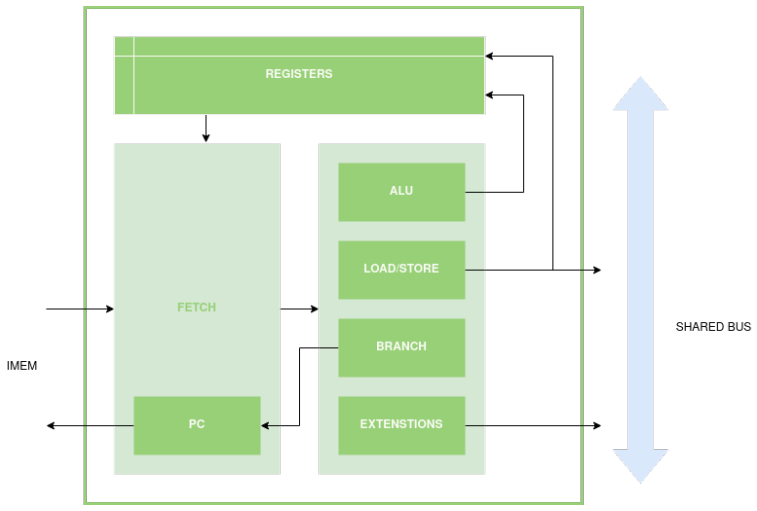


Figure 5. Architecture of a very minimal CPU.

We design a minimal 2 stage pipelined CPU capable of doing basic arithmetic, logic, memory load/store, and custom vector operations. The instruction memory is separated from the data memory. Instructions will be issued in order. For branches, the fetch stage will be flushed, and the program counter will be updated. In the case of vector instructions, the appropriate CSRs of the corresponding accelerators will be updated accordingly.

Auxilary Modules: Memory.

A minimal memory module is designed. It contains addressable blocks with a byte size. The number of read and write ports is parameterized for maximum throughput according to the bus data size.