Lab 4. Pipelined SIMD Processor Implement

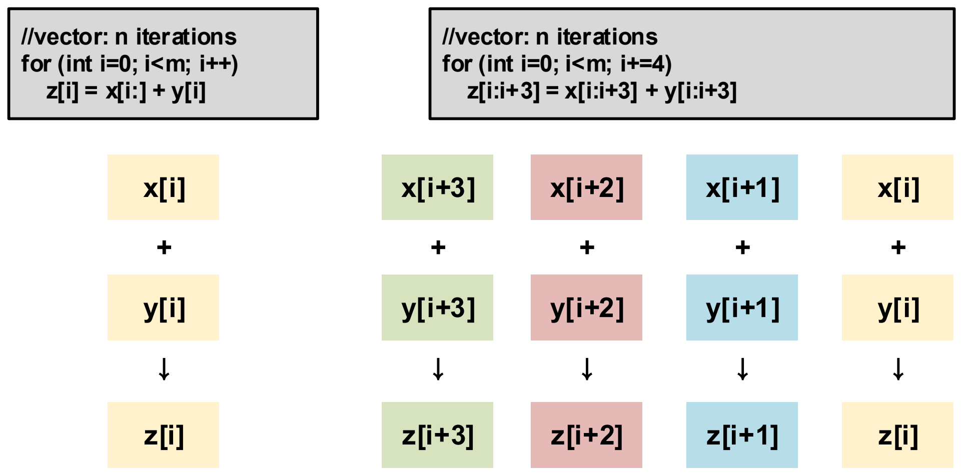
I. Purpose

The purpose of this lab is to design a simple pipelined SIMD processor for dot product. We will implement and synthesis processor by FPGA design tools (Xilinx Vivado & Xilinx Vitis) and FPGA evaluation board (Xilinx Arty A7-35T).

II. Backgrounds

A. SIMD Processor

Single instruction multiple data (SIMD) processor is a type of parallel processing hardware. SIMD processors supports simultaneous computation with multiple data, so exploit data level parallelism. This leads to suitable structure for efficient vector operation processing.

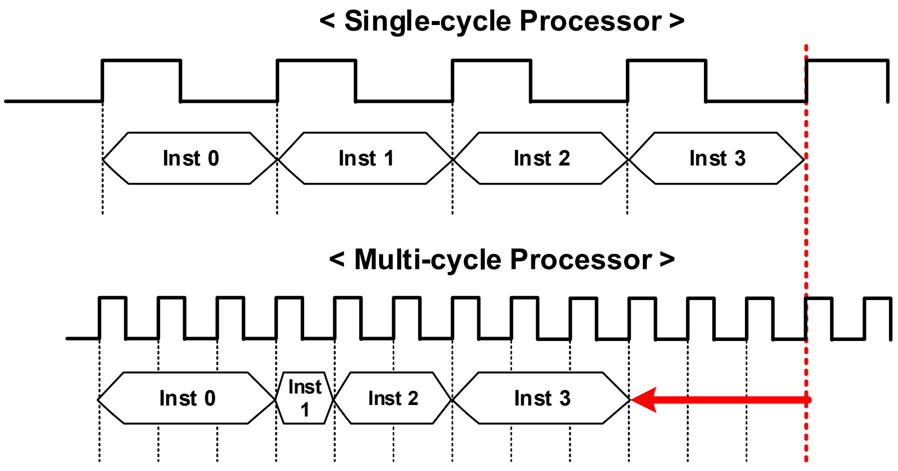


```
//vector: n iterations
for (int i=0; i<m; i++)
   z[i] = x[i:] + y[i]
```

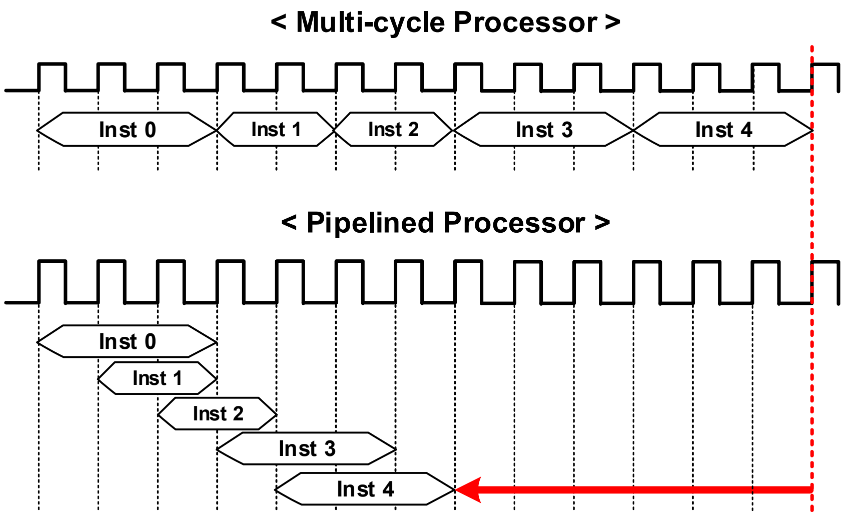
```
//vector: n iterations
for (int i=0; i<m; i+=4)
   z[i:i+3] = x[i:i+3] + y[i:i+3]
```

B. Single/Multi-cycle processor

The single-cycle processor is basic form of processor. Every instruction takes 1 cycle so clock cycle time is determined by slowest instruction. Therefore, it leads to low throughput and long latency. However, in the case of multi-cycle processor with high clock frequency, the number of cycles required for each instruction is different. This optimization shows better performance.

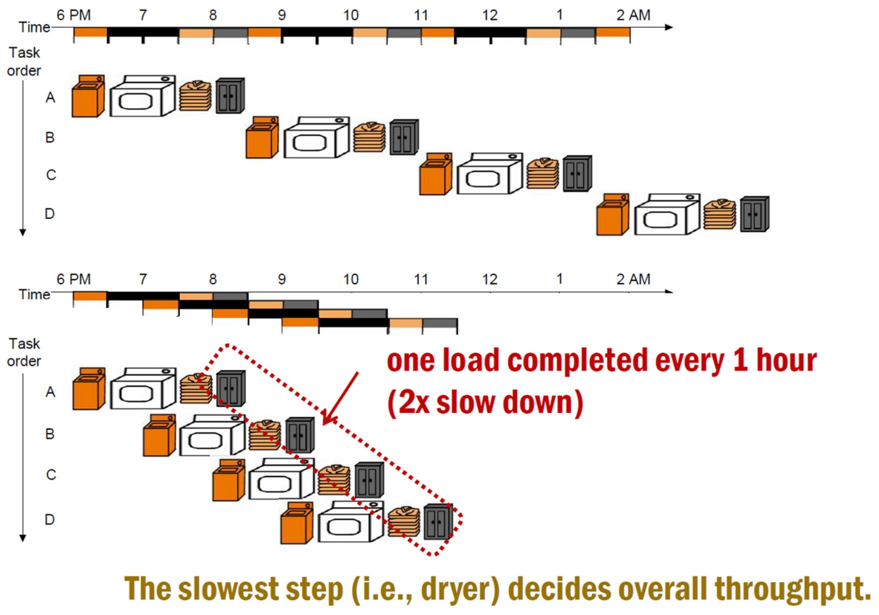


In addition, this structure of multi-cycle processor enables pipelining and presents another way to improved performance.



C. Pipelined Processor

The hardware for running instructions is divided into several stages. For existing processors, only one instruction is processed at the same time. However, pipelining allows each stage to handle different instructions in same time, so increasing the utilization of the hardware. The laundry example below explains the effect of pipelining well.



Time
6 PM
7
8
9
10
11
12
1
2 AM
Task order
A
B
C
D
one load completed every 1 hour (2x slow down)
The slowest step (i.e., dryer) decides overall throughput.

III. Lab Procedure

1. Implement pipelined SIMD processor

i. Prepare golden model (input matrix, vector, output result)

ii. Implement multipliers & adder tree ➔ verify w/ single dot product

iii. Implement pipelined SIMD ➔ verify w/ multiple dot products

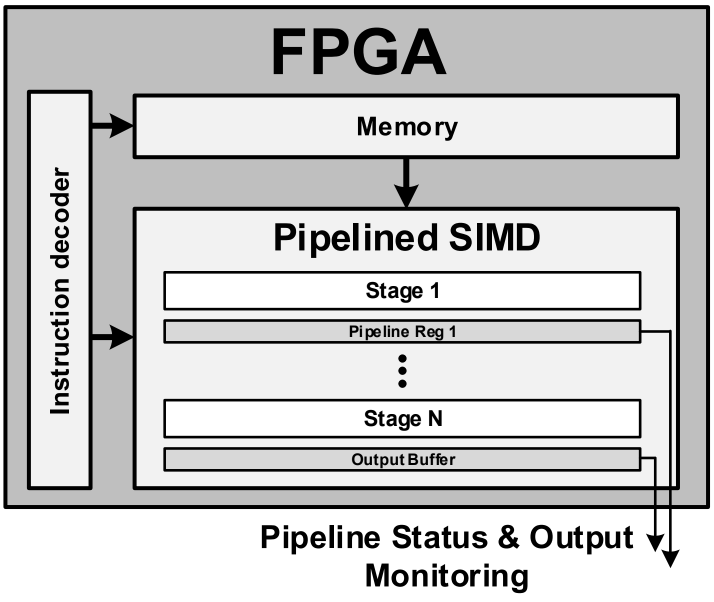


Figure 1. Overall Architecture

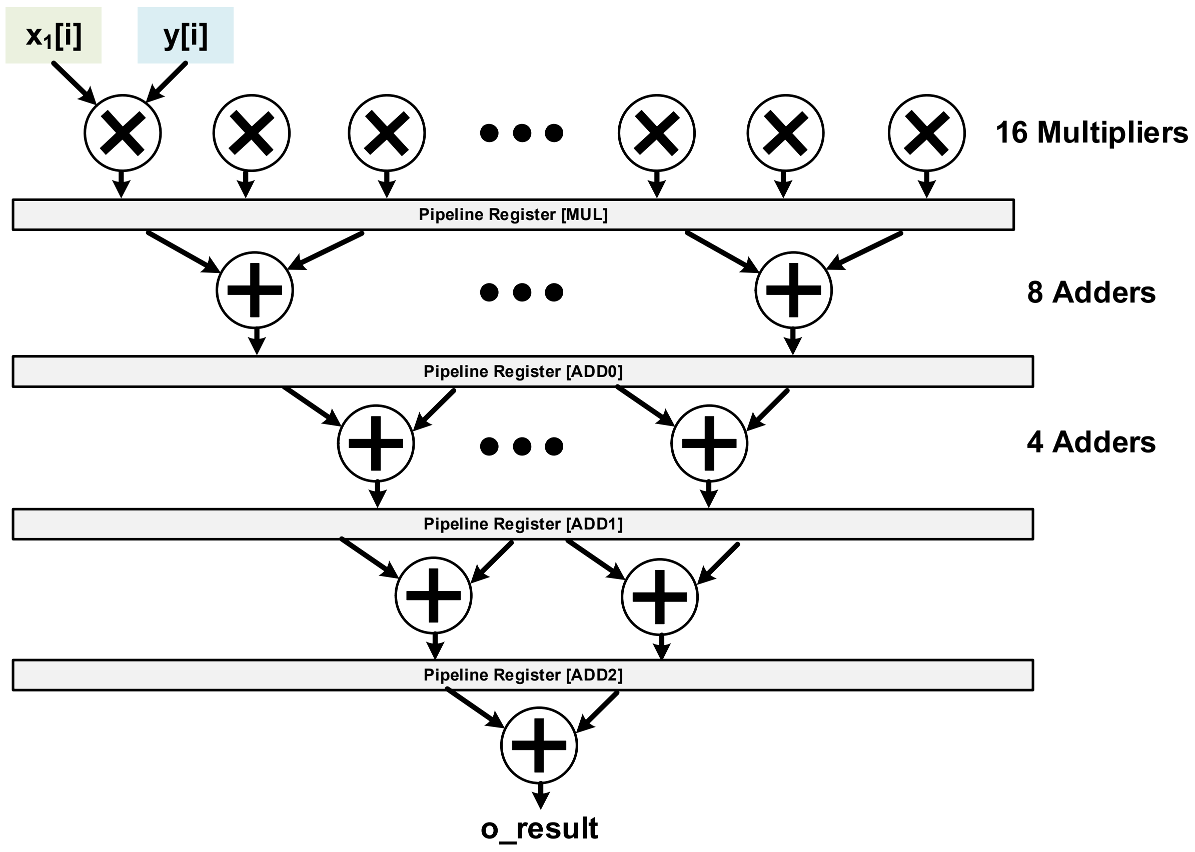


Figure 2. Pipelined SIMD

2. Synthesis processor on FPGA

i. Simulate the full system with the top testbench

ii. Run synthesis and implementation on the FPGA

iii. Run your programmed FPGA with prepared golden vector

iv. Analyze the performance and pipeline reg value
System Performance : Throughput, Latency, System operating clock, FPGA resource

3. Data dependency handling

i. Execute given serial dot product with implemented pipeline processor.

ii. Dot product between operation result required, so data dependency exists.

iii. Sense data dependency of given instructions and solve it with optimized bubble cycles

iv. Analyze the performance and pipeline reg value

IV. Problems

A. Pipelined inner product for 32 input vectors

① Write input/weight data into DUT with instruction

② Run pipelined SIMD for 32 input vectors

③ Write 32 results into output SRAM

④ Call up a new weight every 16 operations

B. Data dependency handling

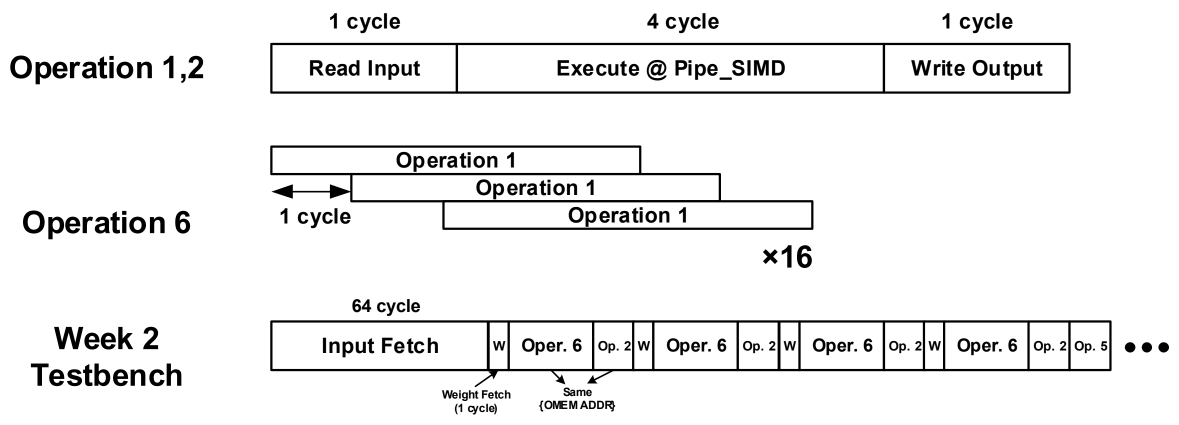
① Write input/weight data into DUT with instruction

② Run pipelined SIMD for 64 input vectors

③ Call up a new weight every 16 operations

④ Perform inner product with output vector every 16 operations

⑤ Control data dependency generated in the process using “stall”.



V. Discussion

A. Is the more stages in the pipeline processor configuration, the better? If not, how can we find the optimal structure?

B. How hardware automatically sense data dependency and provide optimized bubble cycles?