How do you do what you do when you're a z13 CPU

Bob Rogers
IBM Distinguished Engineer, Retired
Trident Services, Inc.; MSP Communications; IBM Corporation

#SHAREorg

SHARE is an independent volunteer-run information technology association that provides **education**, professional **networking** and industry **influence**.

Session objectives

The objective of this session is to give an under-the-covers look as the SMT and SIMD technologies that IBM has introduced on the z13 processor.

Special thanks to my hardware engineering colleagues for teaching me enough to pass this on to the attendees.

- Jonathan Bradbury
- Brian Curran
- Christian Jacobi
- Eric Schwarz
- Kevin Shum

Abstract

The IBM z13 introduces two new technologies to the mainframe: Simultaneous Multithreading (SMT) and Single Instruction Multiple Data (SIMD) processing. In this presentation we take a look under the covers to see how the technologies actually work in the processor code. It also covers some of the many z13 design also refinements on the capabilities of its predecessor, the zEC12.

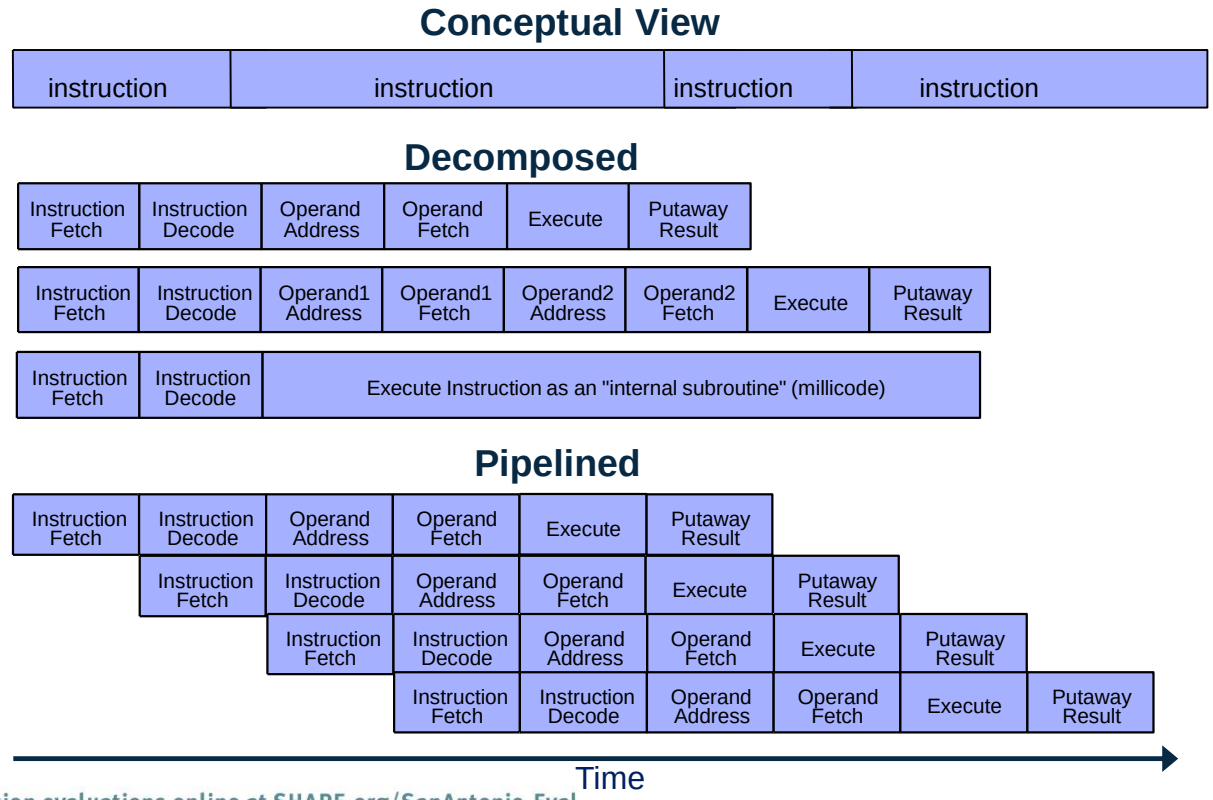
Understanding SMT

Simultaneous Multithreading is a refinement in processor design that is the end of a sequence of ideas that have increased the complexity of the processor to deliver greater throughput.

In order to understand how SMT works and what it means for performance management and capacity planning, it is best to start with how modern computers process instructions.

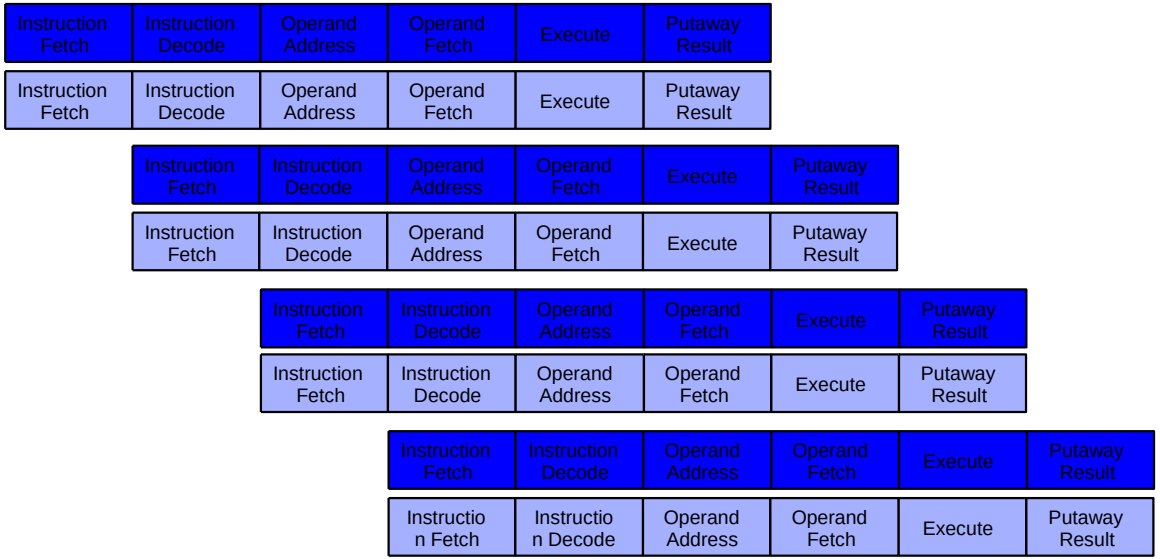
We can start with the evolution of the instruction pipeline.

Conceptual View of Execution vs Reality



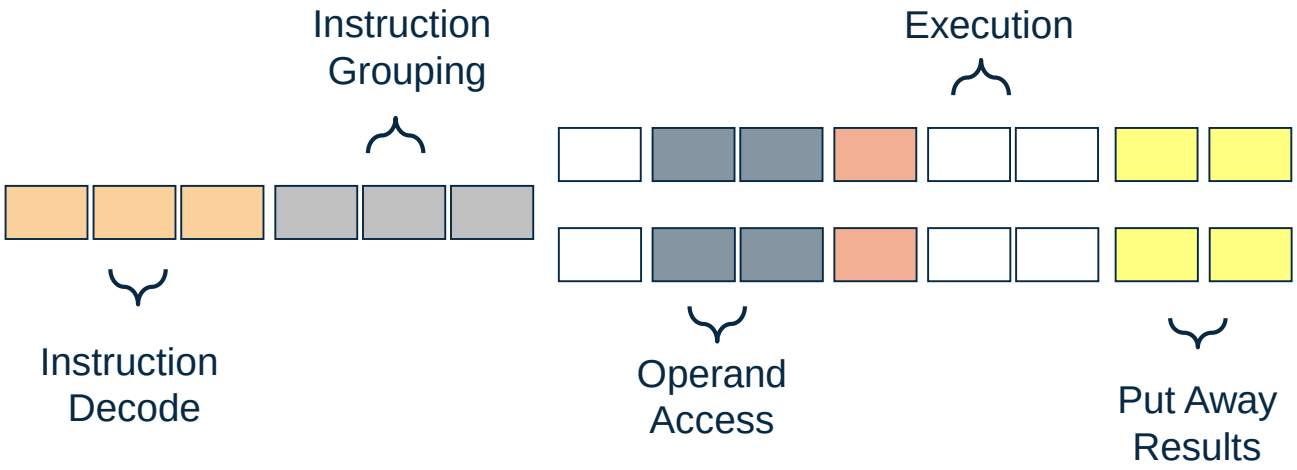
Superscalar: multiple instruction overlap

A Superscalar processor can process multiple instructions simultaneously because it has multiple units for each stage of the pipeline. But, the apparent order of execution is still maintained.



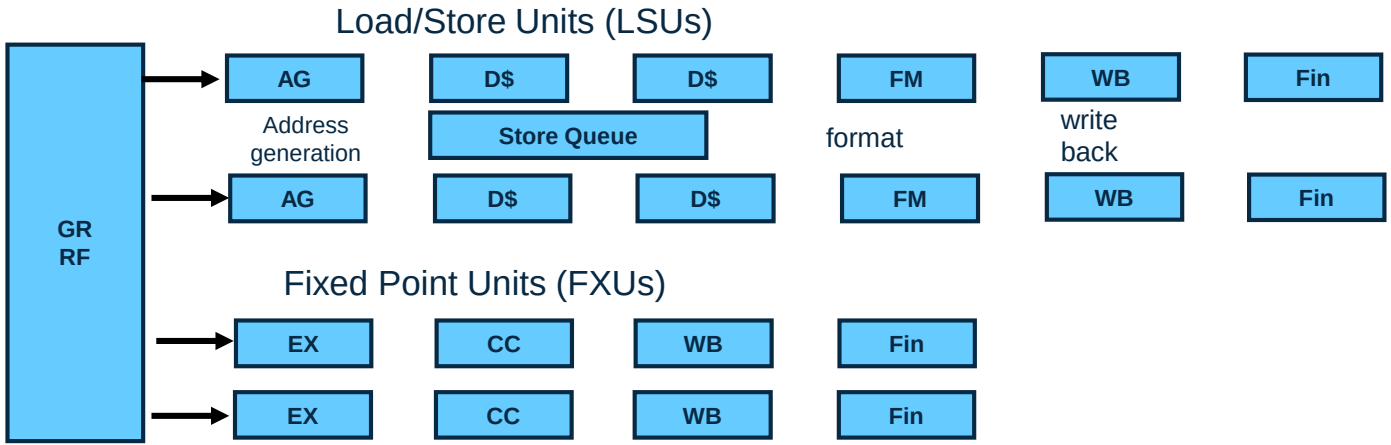
Schematic of Superscalar Pipes

A Superscalar processor can process multiple instructions simultaneously because it has multiple units for each stage of the pipeline.



Each box from left to right represents a pipeline stage which takes one cycle. After instructions have been decoded and put into superscalar groups, they are issued down the two pipes one or two instructions at a time. In this example, the apparent order of execution is still maintained.

RISC-like Superscalar Pipes



Depiction of part of the z196 pipeline showing 2 load/store units and 2 fixed point units. The stages for instruction fetch, decode and grouping and putaway are not shown. Also not shown are the floating point units for binary and decimal arithmetic. Under ideal conditions, the z196 can execute 5 instructions simultaneously.

Out-of-Order Execution

- A processor that can execute instructions Out-of-Order (OOO) uses detailed bookkeeping and some tricks to appear to execute the program as it was written.
- To do the bookkeeping, the processor maintains what is called a global completion table to track the status of all in-flight instructions.
- The results of an instruction cannot be stored until all older instructions have previously completed.
- If, for example, an interrupt occurs, all instructions that have not already stored results must be “forgotten”, and re-executed later. The interruption PSW reflects the newest instruction that stored results (i.e. the last completed instruction such that all preceding instructions had also completed).

Out-of-Order Execution Example

(On a 2-way superscalar processor)

SHARE
EDUCATE • NETWORK • INFLUENCE

Original Instruction Sequence

7 instruction groups and 10 cycles AGI delay

```
seq     instruction text              | seq   instruction text
01  LLGT  @04,XFORNP31                |
02  L     @04,FW(,@04)                | 03  ST    @04,XFORS
04  LG    @05,TOPPTR                  |
05  LG    @09,RTTOP(,@05)             |
06  ST    @04,RSISIZE(,@09)           | 07  SLR   @02,@02
08  ST    @02,RSIPREV(,@09)           | 09  LG    @02,RDIPTR64
10  LH    @08,RDITYPE(,@02)           |
```

Reordered Instruction Sequence

5 instruction groups and 6 cycles AGI delay

```
seq     instruction text              | seq   instruction text
01  LLGT  @04,XFORNP31                | 04  LG    @05,TOPPTR
05  LG    @09,RTTOP(,@05)             | 07  SLR   @02,@02
02  L     @04,FW(,@04)                | 06  ST    @04,RSISIZE(,@09)
08  ST    @02,RSIPREV(,@09)           | 09  LG    @02,RDIPTR64

03  ST    @04,XFORS                   | 10  LH    @08,RDITYPE(,@02)
```

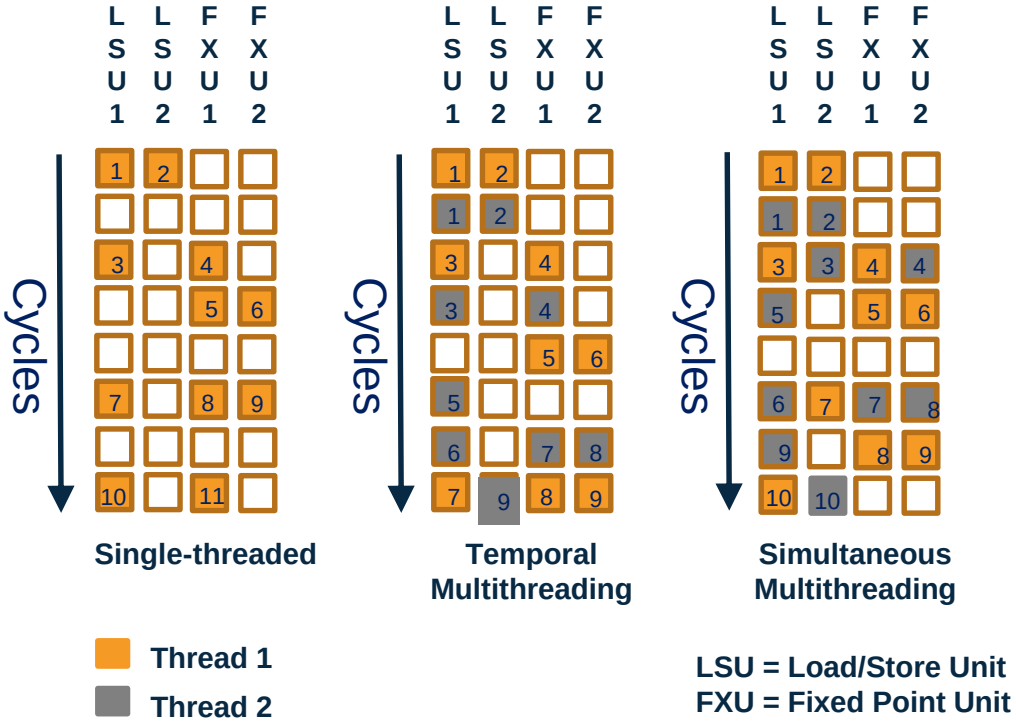
Register Renaming for OOO

- An OOO z/Architecture processor does not have just 16 GPRs.
  - For example, the zEC12 has 80 physical GPRs.
- The reason there are so many physical GPRs is so that the processor can effectively execute instructions out-of-order.
  - The extra physical GPRs allow the processor to reload an architected GPR while its previous value is still needed.
  - Using extra physical registers eliminates GPR interlocks
- Consider this instruction sequence:

```
L    4,0(,1)   Load address of first parameter
L    2,0(,4)   Load first parameter into register 2
L    4,4(,1)   Load address of second parameter
L    3,0(,4)   Load second parameter into register 3
```

- An OOO processor with register renaming can perform both loads of GPR 4 at the same time, and both loads using GPR 4 at the same time.

Single-threaded vs Multithreading



SMT Effectiveness

- Based on some early papers, a core design with two simultaneous threads (SMT2) can deliver an additional 40% throughput.
  - This is a gross approximation and actual results can vary considerably.
  - And, this is only when both threads are busy.
- Unfortunately, the additional throughput from SMT does not scale with the number of threads. This is because all the threads on a core share some limited resources.
- In a case where two threads add (nominally) 40% throughput, a core with four threads (SMT4) can deliver (nominally) only 60% more than single-threaded.
  - That's not even 15% more than SMT2.

Evaluating SMT

- On the positive side, SMT delivers more throughput per core.
  - More capacity for a given footprint size
  - Less power and cooling required per unit of capacity
- But, there are negatives as well.
- The first is that an individual thread in multithread mode is slower than a single thread would be. The speed drops quite rapidly with the number of threads.
  - If an SMT2 core provides 140% of the capacity of a single thread, then two threads will (on average) each run at 70% of the single-thread speed when both threads are active.
  - For SMT4, if all four threads are active, they would run at only 40% of the single thread speed.
- The second negative is an increase in variability.
  - Increased sharing of low-level resources by threads makes the amount of work that a thread can do dependent on what else the core is doing.

What Causes the Slowdown?

- A major cause of less than linear speed-up is the sharing of processor cache.
  - On recent System z processors, there are two levels of cache that are private to the core (on zEC12, they are called L1 and L2).
  - If a core has more than one thread, these caches will be shared across all the threads.
  - Each thread is forces to get by with a smaller footprint in these caches and so takes more L1 and L2 misses than if the caches were not shared.
- Other resources must also be shared:
  - The pipes,
  - The translation lookaside buffer (TLB), and,
  - Physical General Purpose Registers
  - Store Buffers and other resources on the core.

Why the Variability?

- Citing numbers like 140% throughput for SMT2 or 160% throughput for SMT4 are gross simplifications.
- Actual throughput for SMT2 can range from less than 100% (yes, SMT2 can be worse than single-threaded) to close to 200%, depending upon the usage of the shared resources.
- For example, if programs running on the same core stress the same resources, they will run slower than average.
- Alternately, if the programs resource use is complimentary, they can run close to the ideal maximum speed.
- Running the same application multiple times shows less repeatable CPU usage because it may run in differing environments.

What is SIMD?

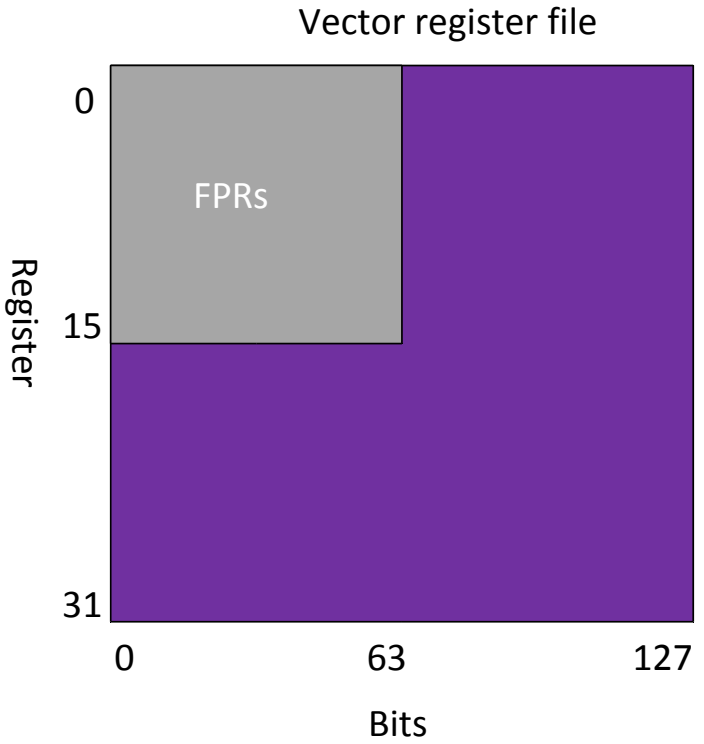
Single-instruction Multiple-data (SIMD) is a technique which uses a single instruction to perform the same operation on multiple sets of operands.

Processor frequencies are not increasing. Even z Systems is stepping back a bit on z13. SIMD provides an additional form of parallelism that can increase overall throughput without requiring an increase in processor frequency.

On z Systems, SIMD is integrated into a traditional (MIMD) processor design. This frees the mainframe from the shortcomings of a fully SIMD Architecture.

How is SIMD implemented on z Systems?

- 32 128-bit registers are defined for vector operations.
- The 16 64-bit floating point registers overlay the leftmost part of the first 16 vector registers.

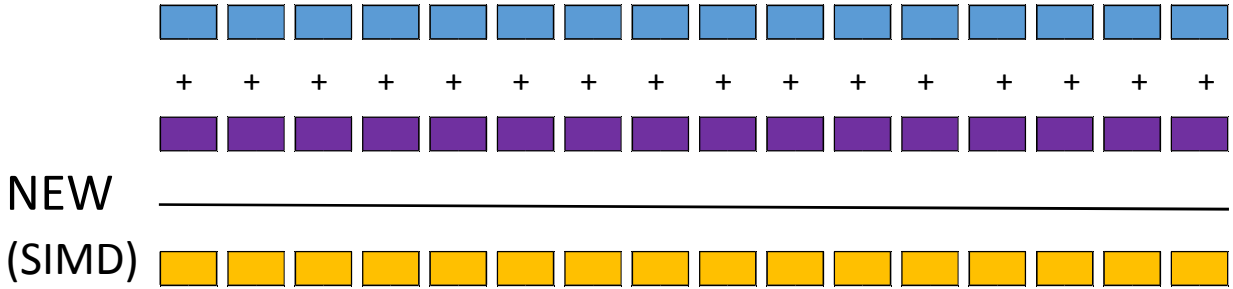


SIMD – Single Instruction Multiple Data

One 64-bit operation

Versus many operations - up to 128 bits

- One 128-bit operation
- Or two 64-bit operations
- Or four 32-bit operations
- Or eight 16-bit operations
- Or sixteen 8-bit operations



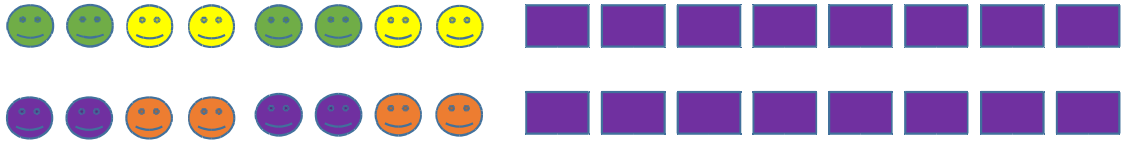
Depiction of sixteen 8-bit additions.

Data Types Supported

- Integers
  - unsigned and two's complement
  - Byte, halfword, word, doubleword, quadword
- Floating-point – z Systems supports more types than any other platform
  - Single Precision (32-bit), Double Precision (64-bit), and Quad Precision (128-bit)
  - Binary Floating-Point (BFP), Decimal (DFP), and Hexadecimal (HFP)
- Character Strings
  - 8-bit, 16-bit, and 32-bit characters
  - Null terminated (C) and Length based (Java)

z13 Floating Point Throughput is Twice as Wide

zEC12 has 1 BFU pipeline



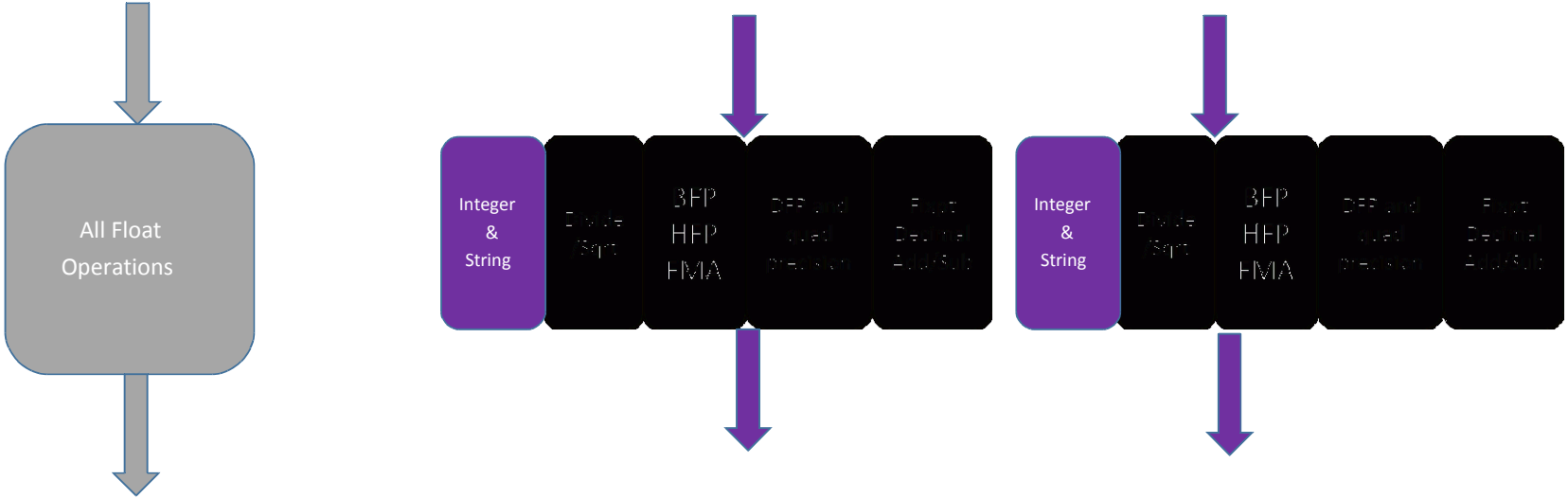
z13 has 2 BFU pipelines

SIMD / Vector architecture makes it easy to clump data To fully utilize 2 BFUs with 1 thread.

Note: a vector operation can be “double pumped” to the same BFU pipeline.

z13 supports additional parallelism and efficiency

z13 allows multiple floating point units to execute at the same time and does not stall on long operations like divide and square root.



zEC12 z13

Integer Vector Instructions

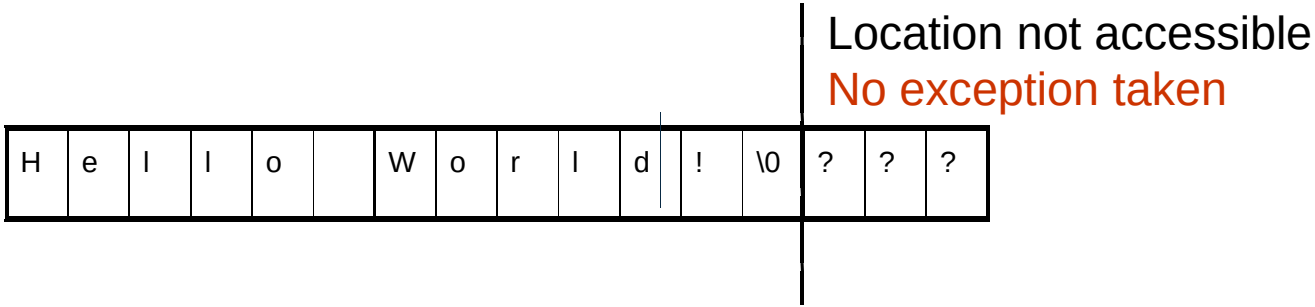
- 8-bit to 128-bit add, sub
- 128-bit add with carry, subtract with carry
- 8-bit to 64-bit min, max, avg, abs, compare
- 8-bit to 32-bit multiply, multiply/add
- Logical operations, shifts,
- Carryless Multiply (8-bit to 64-bit), Checksum (32-bit),
- Memory accesses is efficient even with byte alignment
  - minor penalties for other alignments if they cross a cache line
- Gather by Step

String Vector Instructions

- Find 8-bit, 16-bit, 32-bit, equal or not equal with zero character end
- Range compare
- Find any equal
- Isolate String
- Load to block boundary, load/store with length to avoid access exceptions

Load to Block Boundary Instruction

- VLBB – Loads a VR with as many bytes as it can without crossing a block boundary.
- An immediate field specifies the block size (64B,128B,256B,2K,4K,etc)



Trademarks

Notes:

Performance is in Internal Throughput Rate (ITR) ratio based on measurements and projections using standard IBM benchmarks in a controlled environment. The actual throughput that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput improvements equivalent to the performance ratios stated here.

IBM hardware products are manufactured from new parts, or new and serviceable used parts. Regardless, our warranty terms apply.

All customer examples cited or described in this presentation are presented as illustrations of the manner in which some customers have used IBM products and the results they may have achieved. Actual environmental costs and performance characteristics will vary depending on individual customer configurations and conditions.

This publication was produced in the United States. IBM may not offer the products, services or features discussed in this document in other countries, and the information may be subject to change without notice. Consult your local IBM business contact for information on the product or services available in your area.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Information about non-IBM products is obtained from the manufacturers of those products or their published announcements. IBM has not tested those products and cannot confirm the performance, compatibility, or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Prices subject to change without notice. Contact your IBM representative or Business Partner for the most current pricing in your geography.

This information provides only general descriptions of the types and portions of workloads that are eligible for execution on Specialty Engines (e.g, zIIPs, zAAPs, and IFLs) ("SEs"). IBM authorizes customers to use IBM SE only to execute the processing of Eligible Workloads of specific Programs expressly authorized by IBM as specified in the "Authorized Use Table for IBM Machines" provided at www.ibm.com/systems/support/machine_warranties/machine_code/aut.html ("AUT"). No other workload processing is authorized for execution on an SE. IBM offers SE at a lower price than General Processors/Central Processors because customers are authorized to use SEs only to process certain types and/or amounts of workloads as specified by IBM in the AUT.