New System z Technology: Simultaneous Multithreading

Bob Rogers

Sponsored by The z Exchange and NewEra Software

Abstract

Simultaneous Multithreading (SMT) is a technique for increasing the efficiency of CPUs to deliver more throughput per processor core. It has been exploited on IBM POWER Systems and Intel processors for several years but not yet implemented on IBM System z mainframe processors. However, there have been signals that SMT will be available on the next System z machine. If this happens, we should understand SMT, how it might be implemented on the mainframe, and what it means to mainframe installations. In this webcast, Bob Rogers will provide the historical background and future prognostication to provide insight into this technology before it appears for z/OS and zLinux systems.

Introduction

In my first webcast for NewEra, I spoke of the mainframe's past, present and future. I was prepared, if time permitted, to talk about a new technology that I think is coming to the mainframe. That technology, called Simultaneous Multithreading or SMT, is the topic of this webcast.

It is interesting because it has some of the attributes of a free lunch.

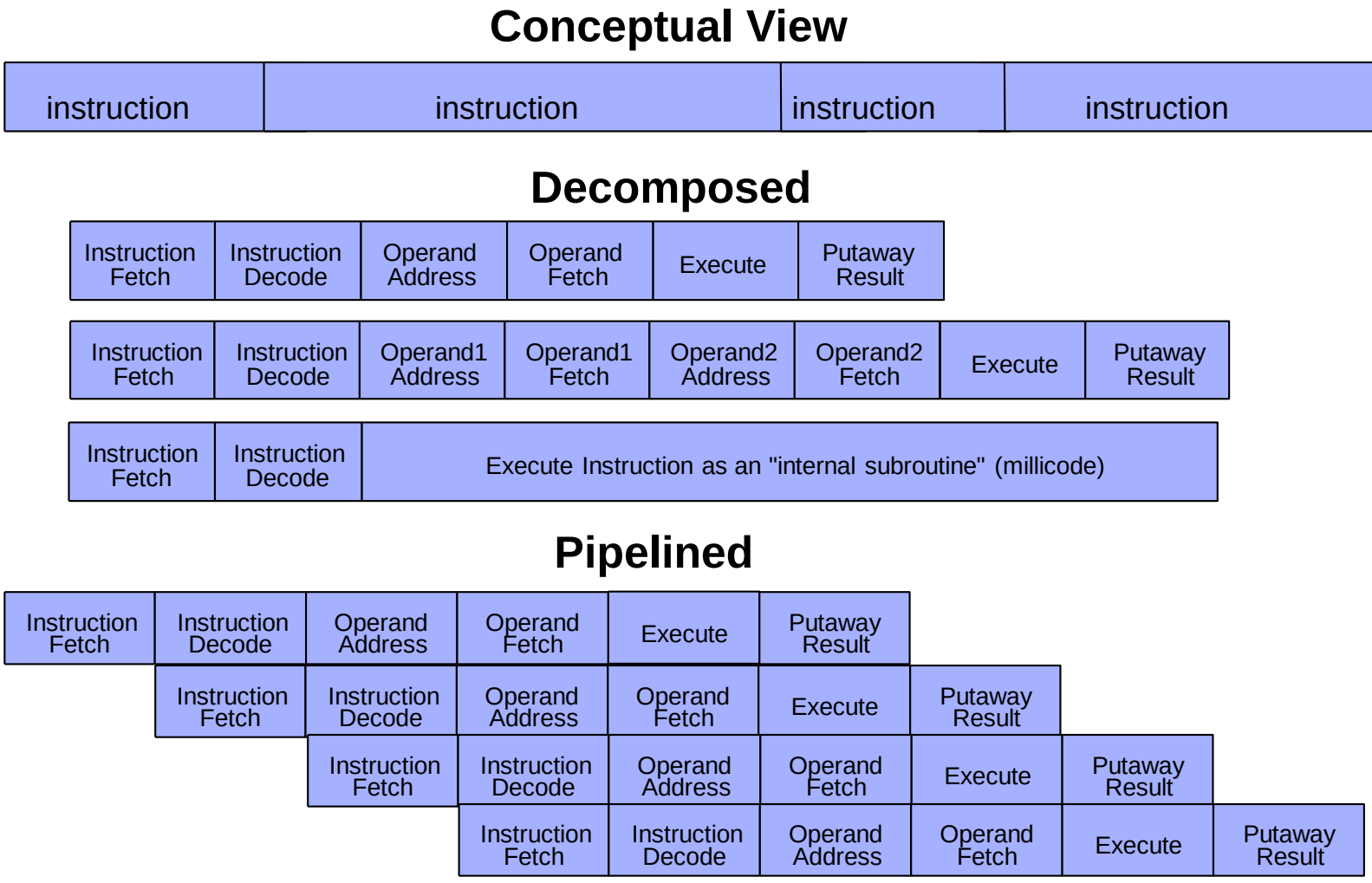
Understanding SMT

Simultaneous Multithreading is 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.

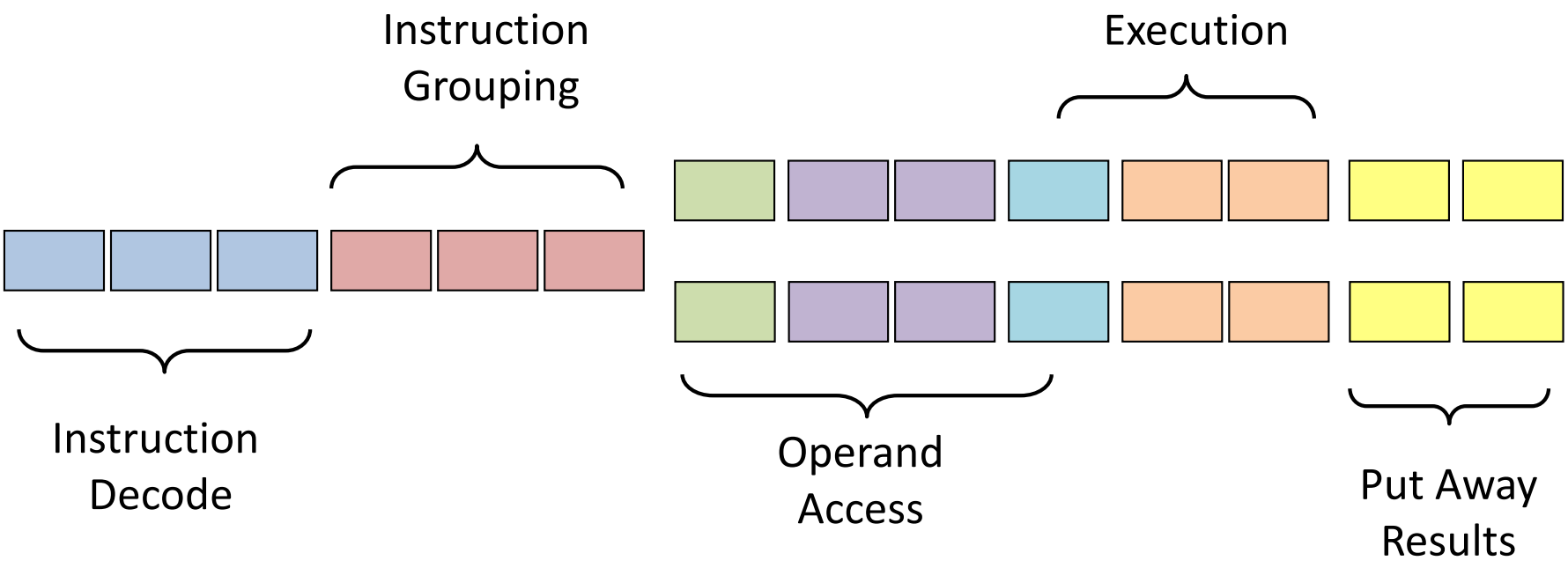
We can start with the evolution of the instruction pipeline.

Conceptual View of Execution vs Reality



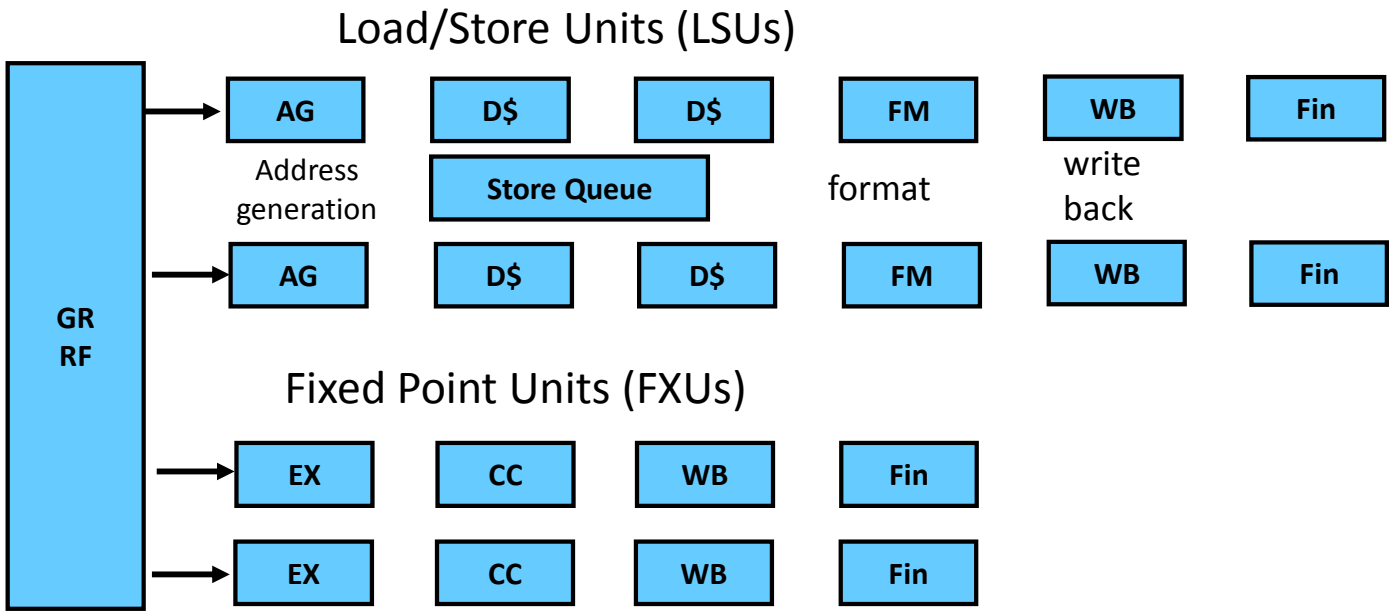
Schematic of Superscalar Pipes

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)

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)
```

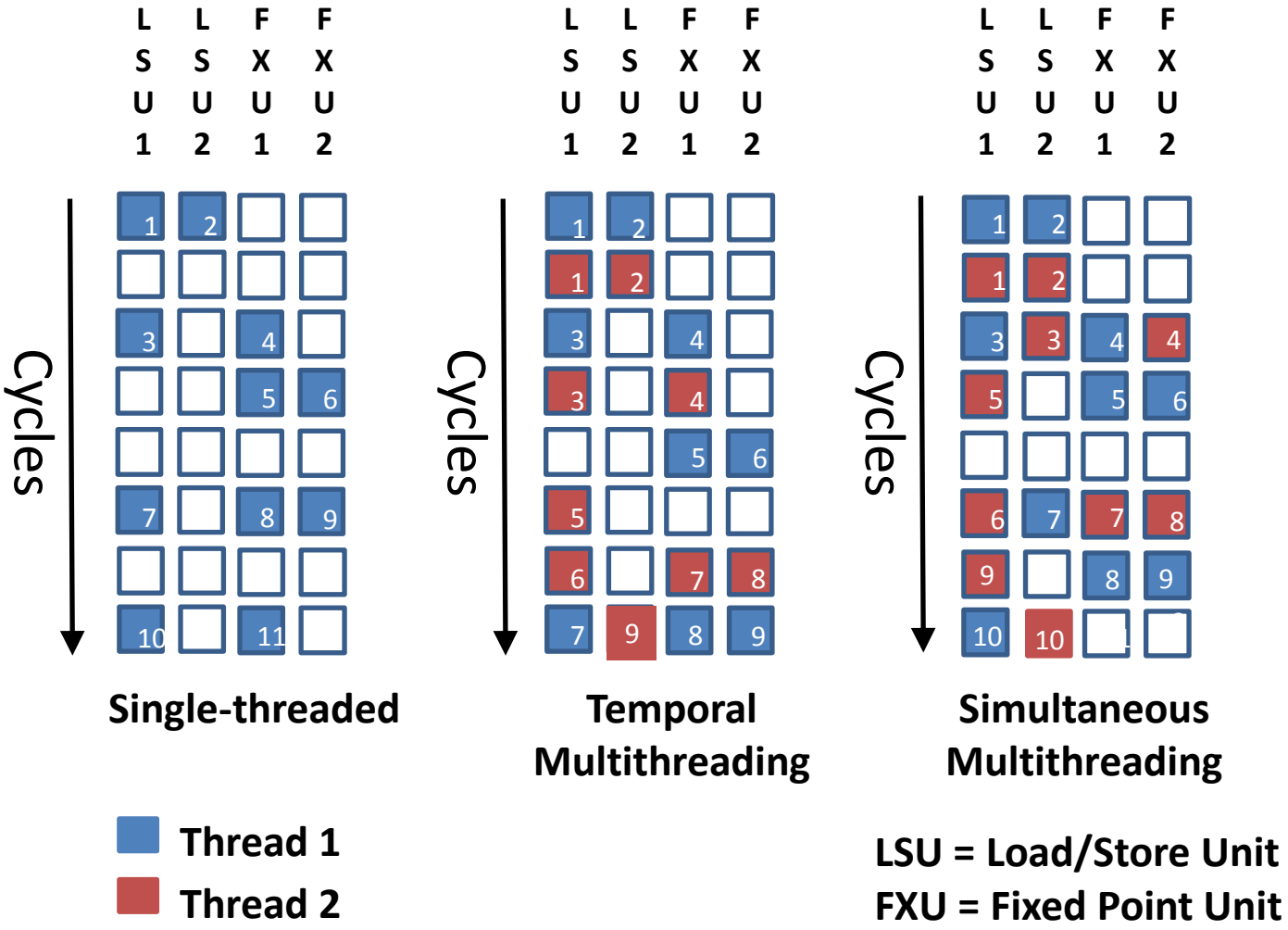
Register Renaming for OOO

- 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.

Some Remediation for Charge-back

- For internal charge-back, IBM has paved the way for more stable charge-back with an enhancement made in z/OS 2.1.
- Even on systems exhibiting wild variations in CPU time for jobs, the number of instructions executed by a job would be expected to be fairly stable.
- To exploit this, z/OS 2.1 can include instruction count data in the SMF type 30 record, provided that the installation has turned on hardware instrumentation (HIS) and told SMF to include the data.
- A charge-back scheme based on instruction counts rather than CPU time might be expected to exhibit more repeatability from one run to the next than one based on CPU time.

Preparing for CPU Slow-down

- z/OS provides fields in the SMF type 30 record that identify the name of the program running on the task that has consumed the most CPU in the time covered by the record and give the percentage of a CPU that that task consumed.
- IBM provides a tool to identify potential problems in a batch flow, called Batch Network Analyzer for z (zBNA).
  - It can be used to explore what-if scenarios by simulating how a batch flow would run on a different speed processor.
- Transaction processing workloads, in general, should not have much problem with the SMT-induced slowdown.
  - The CPU-using portion of a transaction is typically pretty small
  - Furthermore, much of the CPU delay portion of a transaction will be eliminated because the transaction manager will have more threads on which to run the transactions.

Some Prognostication

- The System z hardware will initially support SMT 2 – just like other platforms when they introduced SMT capability.
- PR/SM will not share the threads of a core across multiple guests. All threads will be assigned to a single guest.
- zLinux, when running in a partition, will use the SMT functionality already present for other platform but adapted to the details of z/Architecture.
- zVM will exploit SMT by dispatching guest CPUs on threads without regard to core boundaries.
- Predicting what will be supported by z/OS is more difficult.
  - Supporting SMT for specialty engines would greatly benefit Java
  - The problems of reduced CPU speed and variability are much greater for z/OS than zLinux or zVM.

Questions?

If you are listening on your computer and do not have access to a microphone, please use the Q&A box at the bottom of your screen.

Send your question to Jerry Seefeldt and I will ask it during the Q&A period.

Thank you for attending today's webcast.

You will receive an email with a link to the slide deck used in today's webcast.