Megawhosis and Gigawhatsis?!

Microprocessors Demystified, Transistors Explained and the Increased Importance of Well-written Software Discussed

7 November 2017

David Hutton

STSM, IBM Z Performance and Design

hutton@us.ibm.com

Motivation

- Many folks in the industry understand storage hierarchies while comparatively few understand the microprocessor beyond a “black box” or

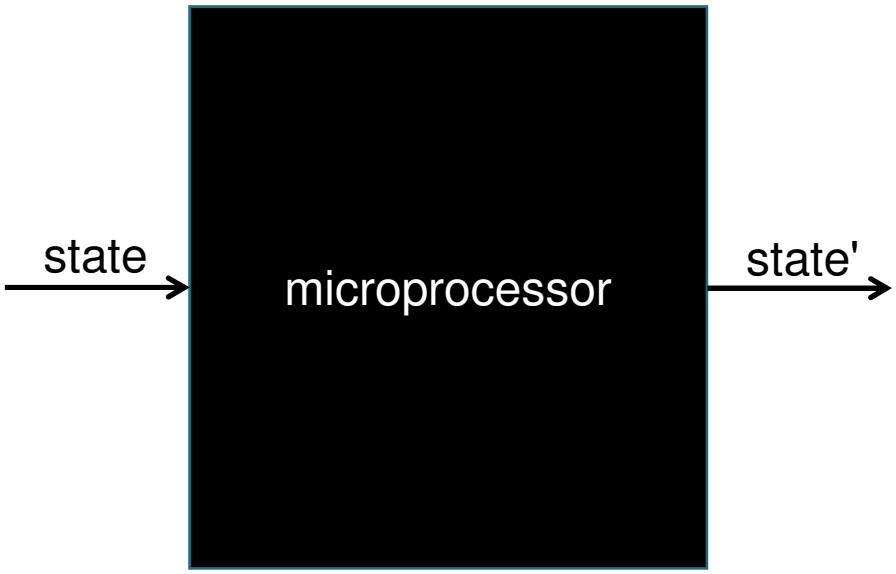
- Many folks rigidly equate higher frequency with higher performance
  - It’s entirely possible that a lower frequency design may outperform a higher frequency design
- With Moore’s Law and Dennard Scaling drawing to a close, dramatic frequency boosts from generation to generation are over, and modern processors deliver continued generational performance through more efficient, wider and clever designs
- Now more than ever, peak value from state of the art hardware can only be attained when it operates upon state of the art software
  - As designs evolve, sometimes tradeoffs are made to jettison logic that enables satisfactory performance around poor programming practices in order to improve performance in the general case
  - Ergo, it’s possible that the same (suboptimal) code will perform worse on a newer generation of processor

Flow of this presentation

- **Section 1: What is a microprocessor? Transistor? VHDL?**
- Section 2: Transistor technology
- Section 3: How technology influences logic
- Section 4: Two design option discussions illustrating the increased importance of well-written software
  - L2 I-and-D-cache balancing: deoptimize the store-into-or-near-the-instruction stream case
  - Store forwarding improvements: deoptimize the rapid, successive store to the same DW case
- Section 5: Summary

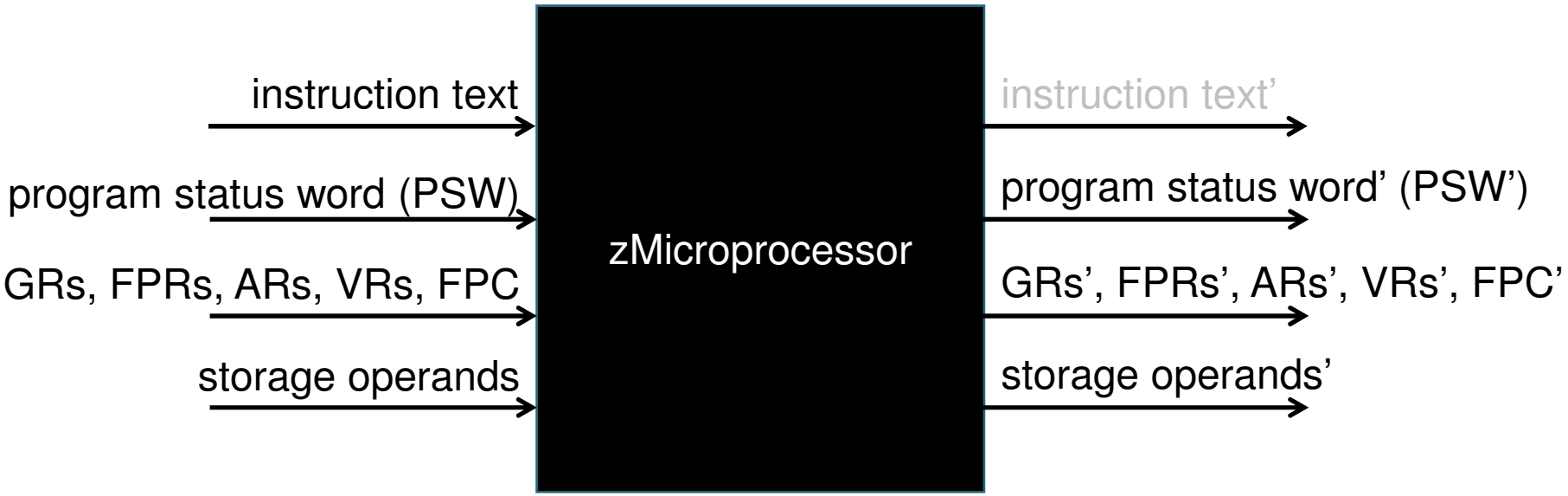
What is a microprocessor?

- A state machine, albeit generally a highly sophisticated one



What is a microprocessor?

- State transitions documented in the zArchitecture “Principles of Operations”
- *Instruction-text modification is possible but not advisable*



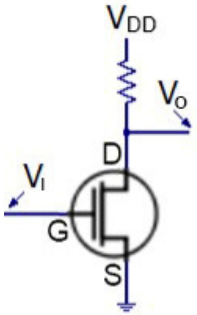
Digital design fundamentals

- What is a transistor?
  - Per Dave Hutton, “it’s a ***voltage-controlled variable resistor”***
  - Analog circuits use them in their full input-output spectrum; e.g., amplifiers
  - Digital circuits use the ends of that spectrum: “on” (saturation region) or “off” only; i.e., binary logic
    - **In digital circuitry, each transistor represents one bit’s-worth of information**

- What’s VHDL?
  - A nested acronym: VLSI (or VHSIC) Hardware Description Language, where VLSI is Very Large Scale Integration (and VHSIC is Very High Speed Integrated Circuit)
  - Can be thought of as “assembler for hardware design”, where the statements are executed in **parallel** rather than **sequentially**

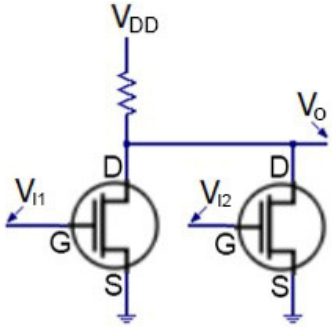
Binary MOSFET logic circuits and truth tables

Inverter



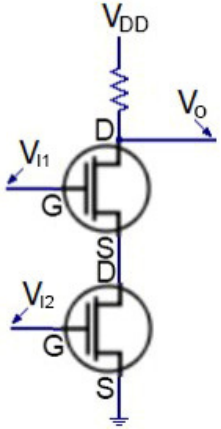
I	O
0	**1**
1	**0**

NOR
(negative OR)



I1	I2	O
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

NAND
(negative AND)

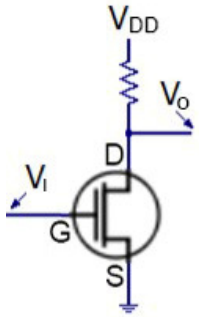


I1	I2	O
0	0	**1**
0	1	**1**
1	0	**1**
1	1	**0**

'I'nput = 0 opens the electrical switch, closes the mechanical gate
'I'nput = 1 closes the switch, opens the mechanical gate

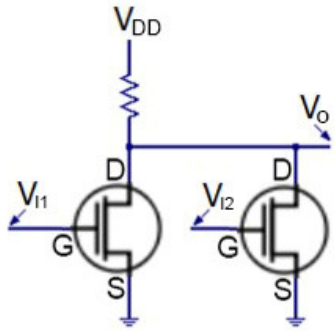
Binary MOSFET logic circuits and truth tables

Inverter



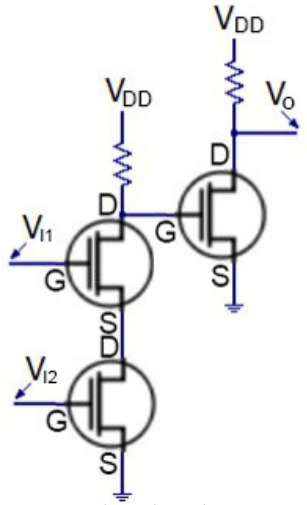
I	O
0	**1**
1	**0**

NOR
(negative OR)



I1	I2	O
0	0	**1**
0	1	**0**
1	0	**0**
1	1	**0**

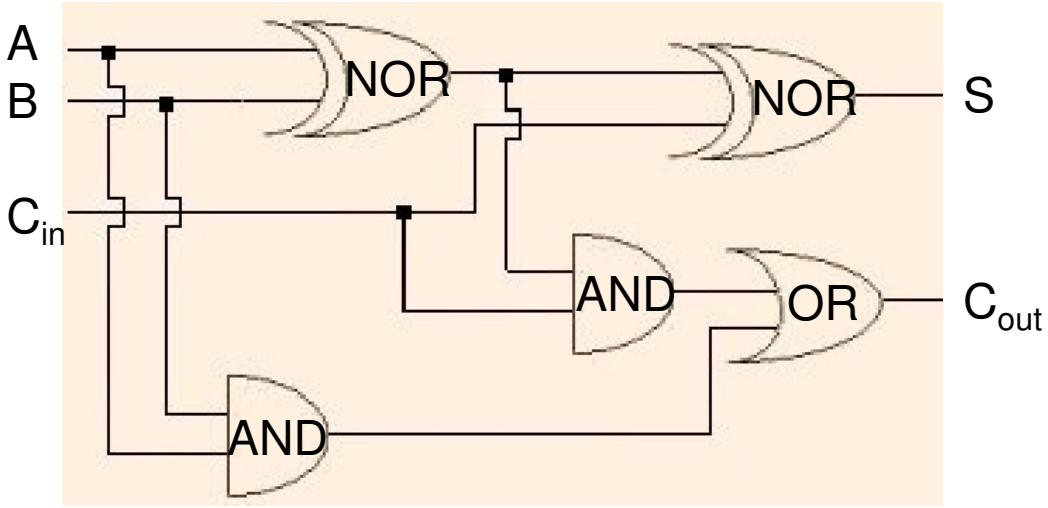
AND



I1	I2	O
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

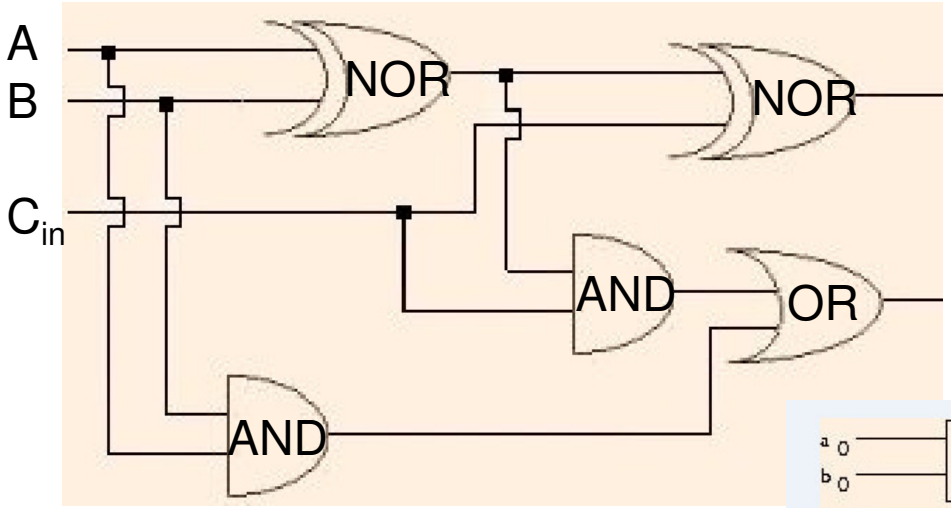
'I'nput = 0 opens the switch
'I'nput = 1 closes the switch

Example 2-input full adder circuit and truth table

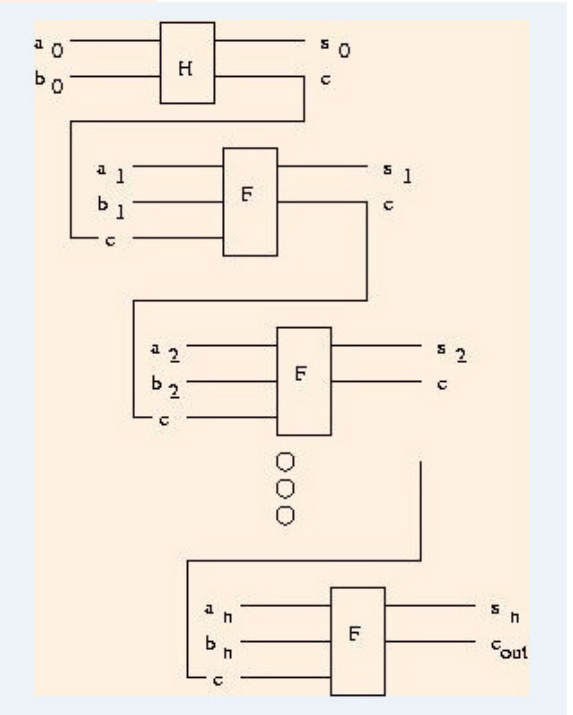


A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

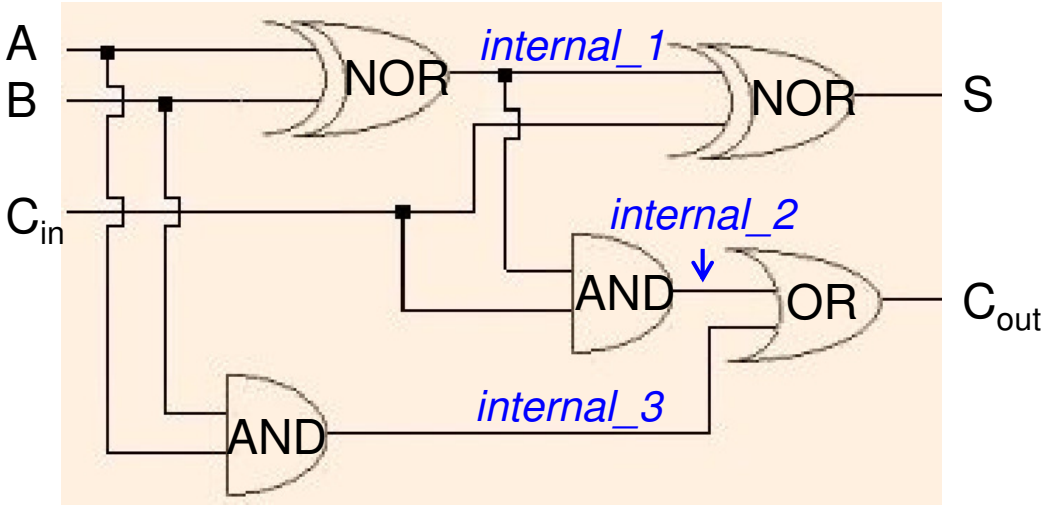
Example 2-input full adder circuit and truth table



A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



Example 2-input full adder circuit and VHDL

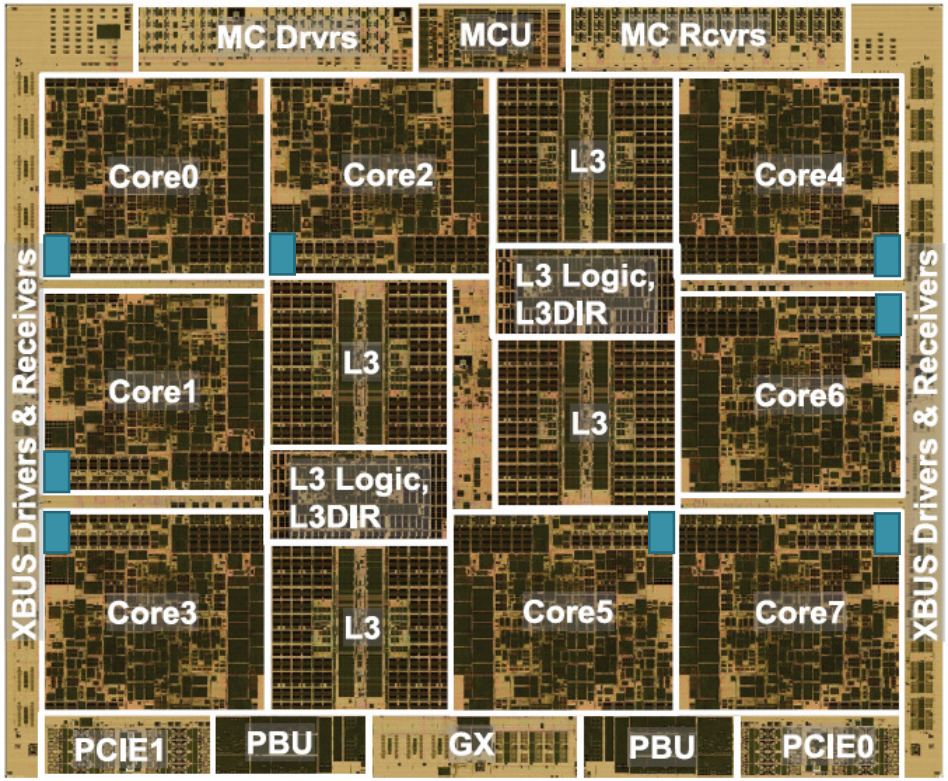



```
-------------------------------------------------
--  FULL ADDER VHDL
-------------------------------------------------

library ieee;
use ieee.std_logic_1164.all;

Entity full_adder_entity IS PORT (
  input_a  : in  std_logic;
  input_b  : in  std_logic;
  input_c  : in  std_logic;
  output_s : out std_logic;
  output_c : out std_logic;
);

Architecture full_adder of full_adder_entity is

  SIGNAL internal_1 : std_logic;
  SIGNAL internal_2 : std_logic;
  SIGNAL internal_3 : std_logic;

begin

  internal_1 <= input_a    NOR input_b;
  internal_2 <= internal_1 AND input_c;
  internal_3 <= input_a    AND input_b;

  output_s <=   internal_1 NOR input_c;
  output_c <=   internal_2 OR  internal_3;

end Architecture full_adder;
```

Flow of this presentation

- Section 1: What is a microprocessor? Transistor? VHDL?
- **Section 2: Transistor technology**
- Section 3: How technology influences logic
- Section 4: Two design option discussions illustrating the increased importance of well-written software
  - L2 I-and-D-cache balancing: deoptimize the store-into-or-near-the-instruction stream case
  - Store forwarding improvements: deoptimize the rapid, successive store to the same DW case
- Section 5: Summary

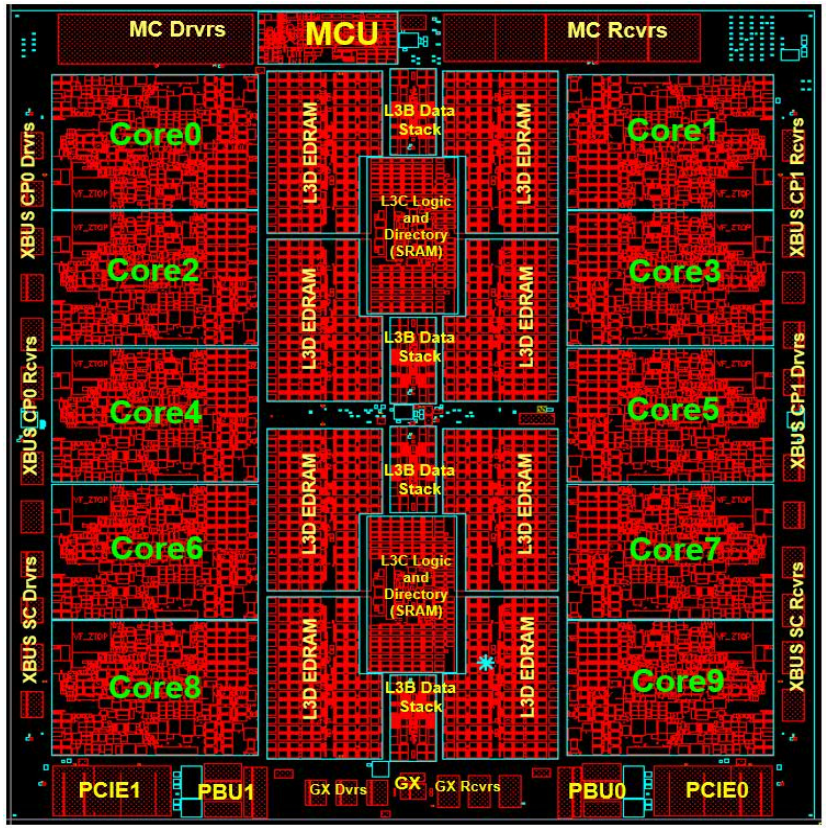
z13/z13s (2964/2965-series) core chip spec sheet – 2015



- **22 nm technology**
- **17 layers of metal**
- **3.99 billion transistors**
- **22 km or 13.7 miles of wire**
- **28.4 mm x 23.9 mm, or 1.2 in x .94 in**

- Up to eight active cores per chip
- 5.0 GHz (9% down from zEC12's 5.5 GHz)
- L1 cache per core: 96 KB I$, 128 KB D$
- L2 cache per core: 2MB L2-I$ + 2MB L2-D$ eDRAM (and integrated i-and-d L1$/L2$ designs)
- L3 cache per chip
  - 64 MB (2 banks @ 32 MB each)
- Single Instruction, Multiple Data (SIMD)
- Supports both single threaded (ST) plus 2-way simultaneous multithreading (SMT2)
- Improved instruction bandwidth
  - Improved branch prediction
  - Double-wide Decode, Dispatch and Completion: increased to 6 uops/cycle (up from 3 on zEC12)
  - Execution: increased to 10 uops/cycle (up from 8 on zEC12)

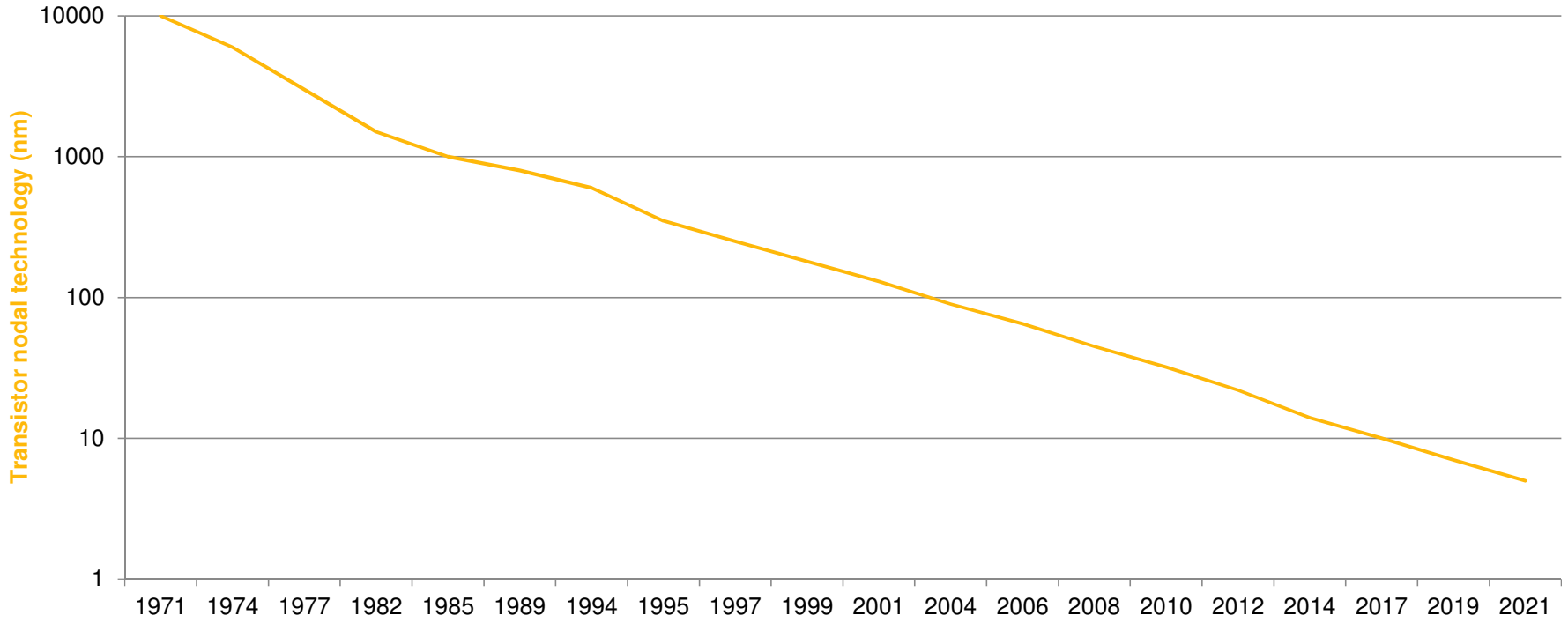
z14/z14s (3906/3907-series) core chip spec sheet – 2017



- **14 nm technology**
- **17 layers of metal**
- **6.14 billion transistors**
- **23.2 km or 14.4 miles of wire**
- **25.3 mm x 27.5 mm, or 1 in x 1.1 in**

- Up to ten active cores per chip
- 5.2 GHz (4% up from z13's 5.0 GHz)
- L1 cache per core: 128 KB I$, 128 KB D$
- L2 cache per core
  - 2MB L2-I$ + 4MB L2-D$ eDRAM
- L3 cache per chip
  - 128 MB (2 banks @ 64 MB each)
- Second generation SIMD
  - Decimal architecture support
- Second generation SMT2
  - Improved thread fairness
- Improved instruction bandwidth
  - Improved branch prediction
  - Enhanced out-of-order issue/execution
  - Hot cache line tempering:
    - Hot line table
    - XI strong-arming

Some perspective on the scaling of semiconductor technology

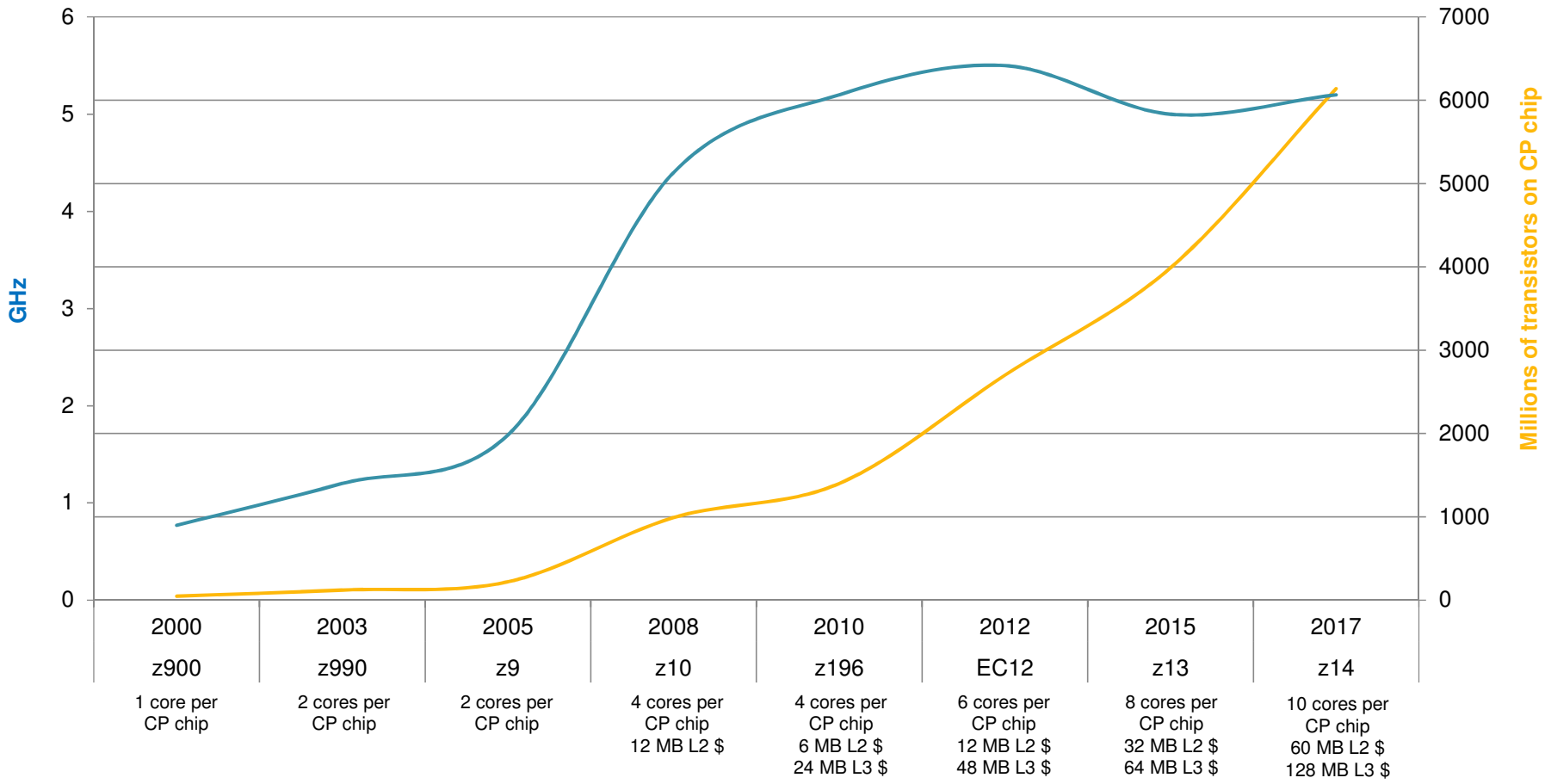


For reference:

- 1 nm = one billionth of a meter, or 0.000 000 001 m, or 10 Å
- a silicon atom's diameter is 2.1 Å or .21 nm

IBM Z processor chip frequency and transistor density

Note: all of the CP chips are roughly the same physical size



Moore's Law and Dennard Scaling

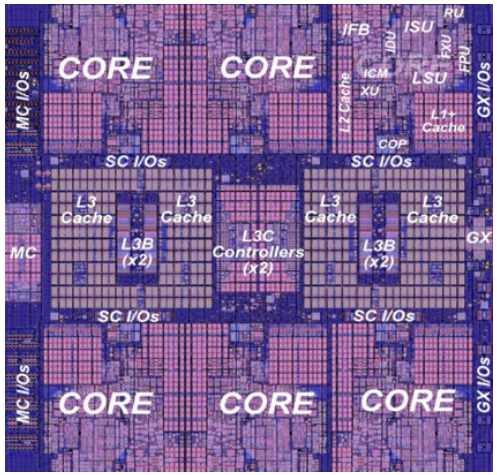
- Moore's Law is a **forecast** that transistor density doubles roughly every 2.5 years
  - Originally proposed in a 1965 paper as doubling every 18 months by Gordon Moore who co-founded Fairchild Semiconductor and Intel
  - Opinions vary, but Moore's Law will likely come to an end (read: **NOT DEAD YET!**) within the next decade-*ish* due to the limits of classical physics
    - *…did I mention transistors are approaching the width of a silicon atom?!*
- Dennard Scaling is a **rule** that a transistor's power consumption scales with its size
  - Originally proposed in a 1974 MOSFET scaling paper co-authored by Robert H. Dennard, who invented Dynamic Random Access Memory (DRAM) and is an IBM Fellow Emeritus
  - Restated, the rule says the power consumption of transistors in the same area of silicon will be roughly the same over time despite their density logarithmically increasing
  - Dennard Scaling has already mostly hit the wall, and again due to the limits of classical physics; e.g., smaller transistors have thinner walls and therefore **higher leakage currents**
- The result: microprocessor frequencies are stagnating while transistor densities continue to increase (the "power-density barrier" is being hit)
  - Restated, **while more transistors per unit size are still available for design opportunity, they're not getting any faster – *and it's no accident SMT is now "a thing"***

Flow of this presentation

- Section 1: What is a microprocessor? Transistor? VHDL?
- Section 2: Transistor technology
- **Section 3: How technology influences logic**
- Section 4: Two design option discussions illustrating the increased importance of well-written software
  - L2 I-and-D-cache balancing: deoptimize the store-into-or-near-the-instruction stream case
  - Store forwarding improvements: deoptimize the rapid, successive store to the same DW case
- Section 5: Summary

4.77 MHz to 5.5 GHz – what is “clock frequency”?

- A common attribute of a microprocessor, this is how many times its “master clock” fires per second
- A better way to think about clock frequency is by its inverse, or “**cycle time**”, expressed in (nano)seconds
  - 4.77 MHz = **209 nanosec** (Intel 8088 released in 1979 in the original IBM desktop PC)
  - 5.5 GHz = 5500 MHz = **0.182 nanosec** (IBM zEC12 released in 2012*), whose frequency is **1153x the 8088’s!***
- **Cycle time represents the maximum allowable wall clock time that signals have to leave every “latch” output, *percolate along wires and through (transistor-constructed) logic devices*, and arrive-and-stabilize on the latch inputs at the next pipeline stage**
  - A “latch” is a transistor construct with an input, an output, a clock and a constant source voltage, such that at the moment the clock pulse rises (or falls) the input is reflected on the output and remains stable until the next clock pulse; stability is maintained by the source voltage
- When the clock fires, each latch’s input is reflected on its output, and the next pipeline stage of an instruction’s work may proceed



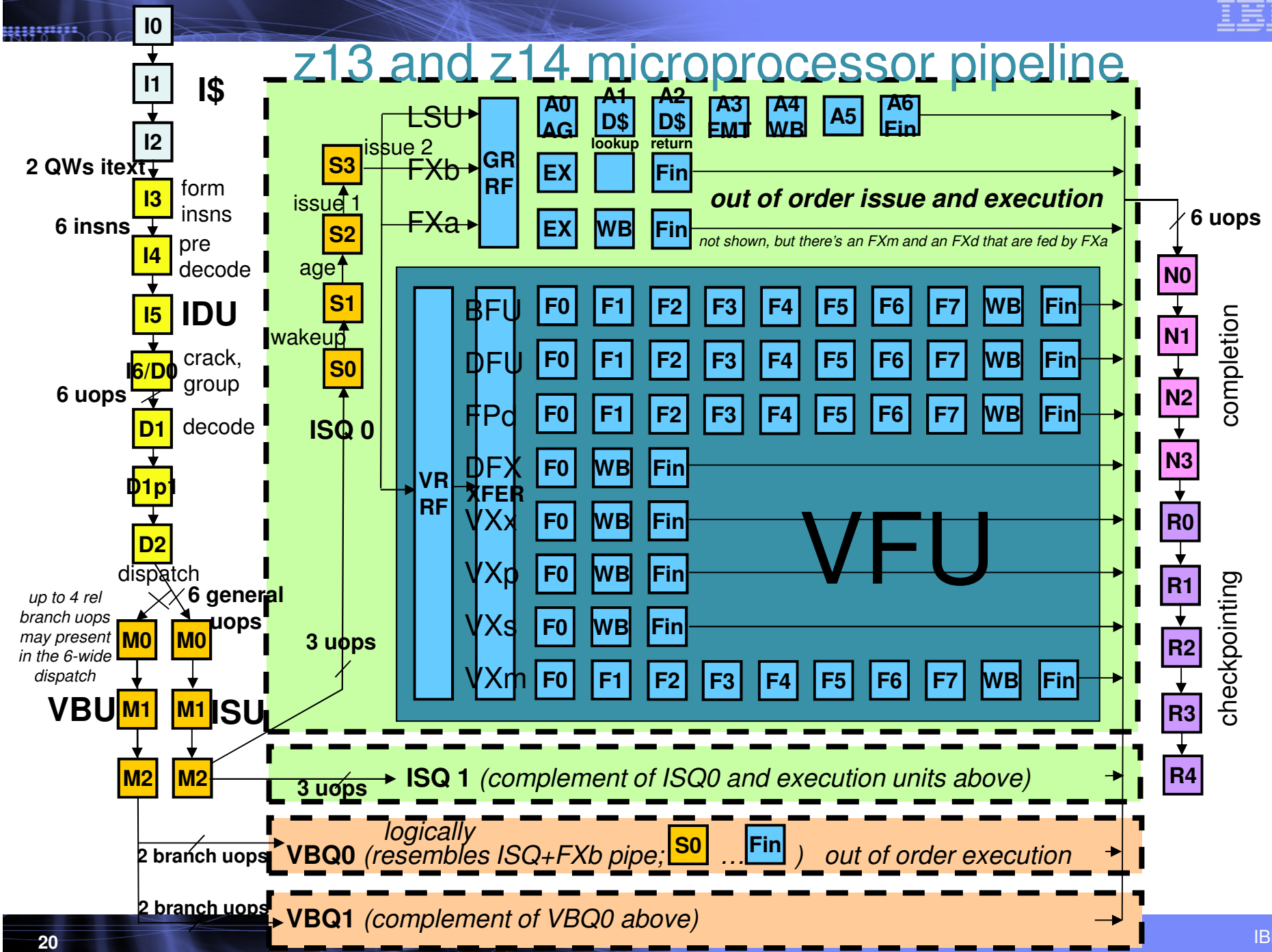
zEC12 6-core CP chip
(roughly 1 in squared)

The red line is the approximate distance light travels in a vacuum within one zEC12 clock cycle (~55 mm or 2.15 in)

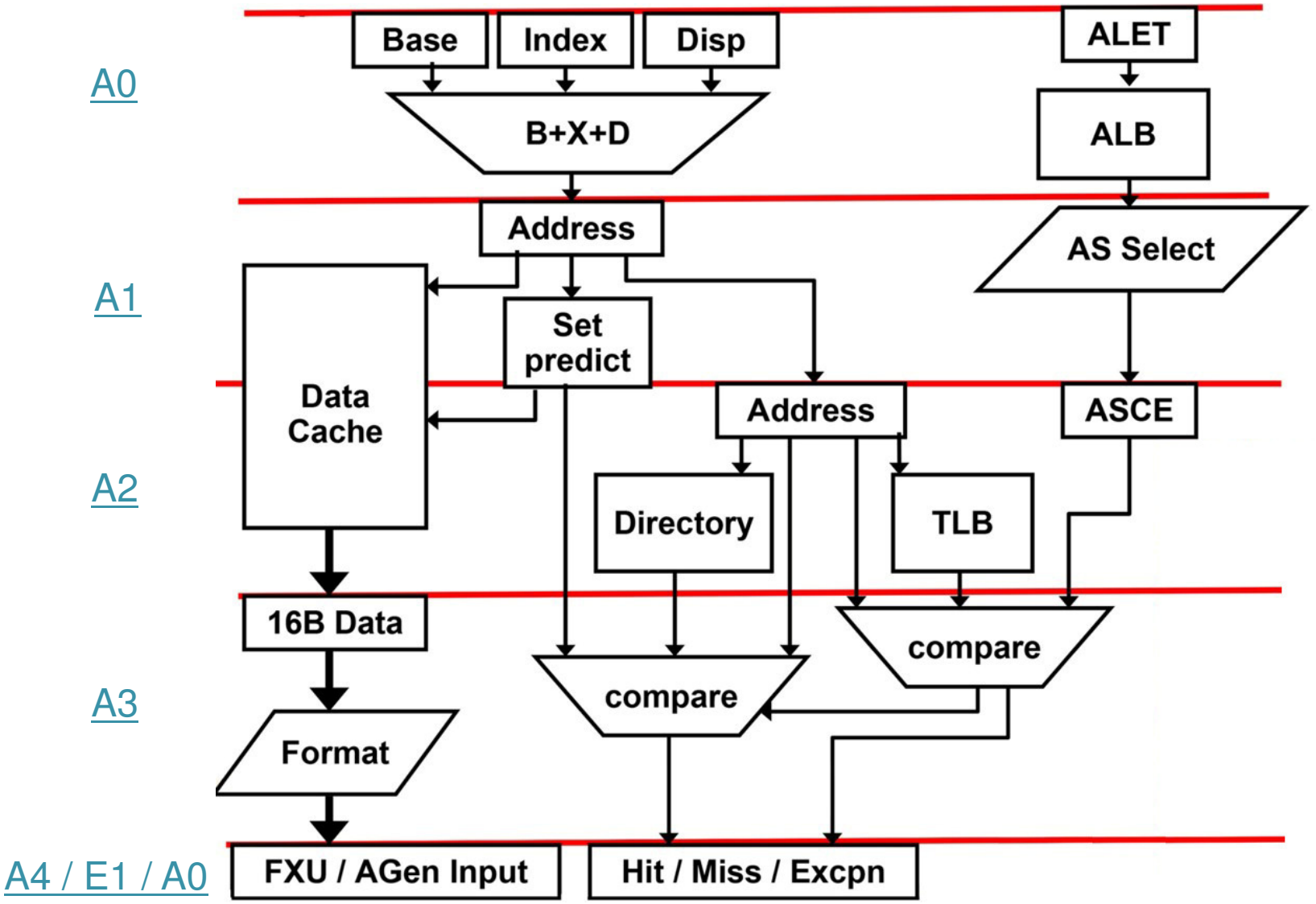
*That is, if a “light year” is 3.725 x 10^{17} or 372 500 000 000 000 000 inches, then a “light zEC12-clock-cycle” is **2.15 inches***

The green line is the approximate achievable transmission reach in one zEC12 clock cycle (~2.5mm or .1 in)

z13 and z14 microprocessor pipeline



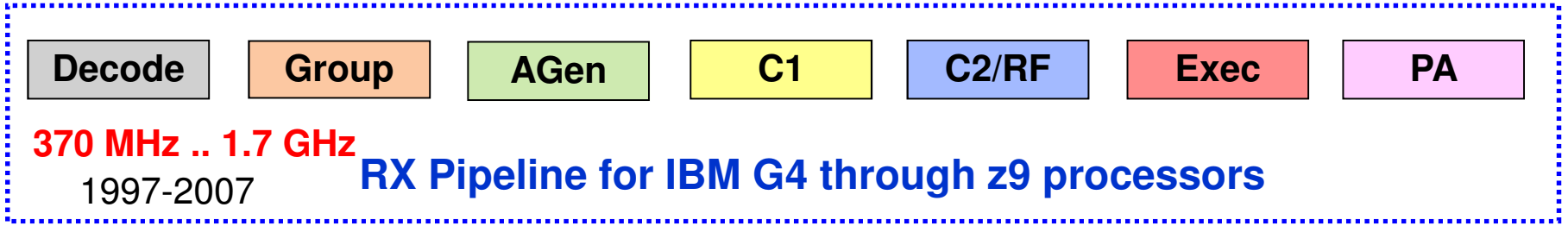
High frequency pipelining example: z10..z14 operand access path

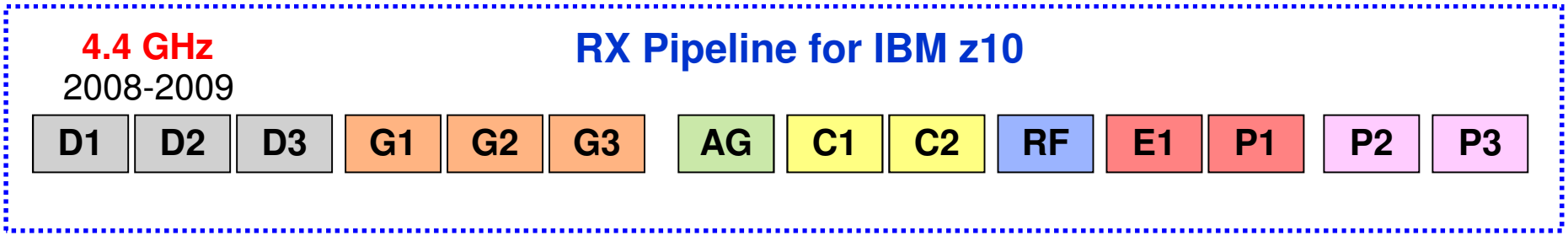


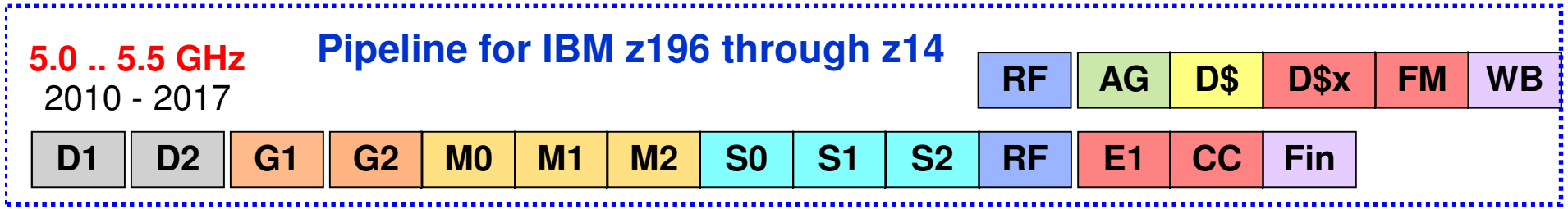
Higher frequency means higher performance, right?

- ***Not necessarily***
- Higher frequencies translate into deeper pipelines and smaller caches
  - At higher frequencies, the pipeline stages (transistor-based logic circuitry between latches) within the pipeline have less time to do real work, therefore more stages are required to do the same total volume of work
    - This complicates stalling logic resulting in overdetection and “recycling” mechanisms
    - Pipeline flushes due to mispredicted branches become more costly
    - Some single-cycle instructions become multi-cycle instructions
  - A microprocessor design-performance principle is that *the Level 1 cache(s) must be accessible in a single cycle*, and **the larger the cache, the more access time required**
    - e.g., cycle X: index the cache, cycle X+1: data returned from the cache
    - If this principle is violated, “bubbles” are injected on every cache access which prevents back-to-back execution and impacts performance
  - Cache and memory hierarchy longer in terms of absolute cycles
- However… by rebalancing your design, you can net yourself a win
  - Smarter branch prediction logic, smarter prefetching logic, extra cache layers…

Simplified IBM Z core pipelines: G4 through z14

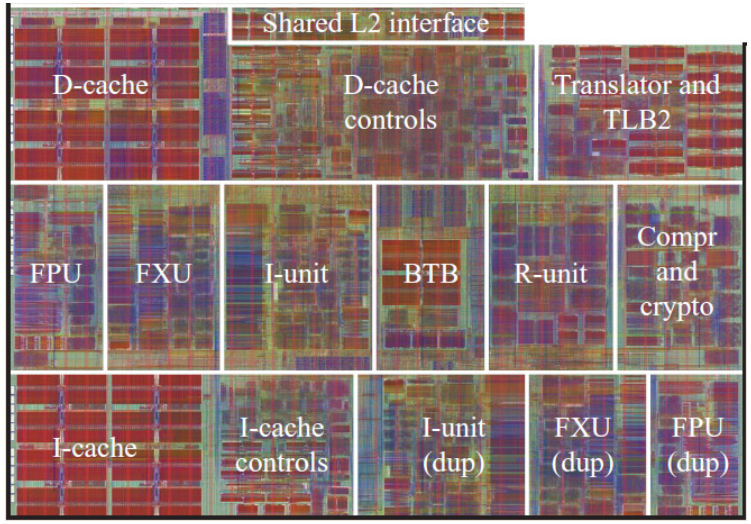






ISU mapping and dep analysis for **out of order execution**
Split inline-RX pipeline into separate load/store and arithmetic uops (SIMD/BFP/DFP/VBU not shown)

z990 vs z13 individual core floor plan – *shown to approximate scale*



Feature	z990 core	z13 core
Announce year	2003	2015
L1$	256 KB iL1$ + 256 KB dL1$	96 KB iL1$ + 128 KB dL1$
L2$	0 MB	**2 MB iL2$ + 2 MB dL2$**

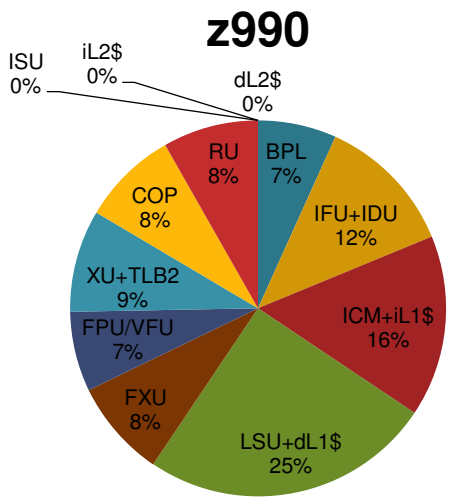
*That is z13's L1 $ is 44% of z990's, however its total on core cache is ~**8.4x** larger than z990's*

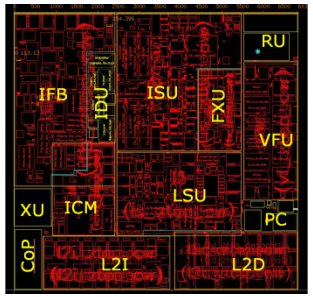
Decode/dispatch width	2-wide	6-wide w/ cracking
Execution	in-order RX	**out-of-order (ISU)**
Execution width	3-wide	10-wide

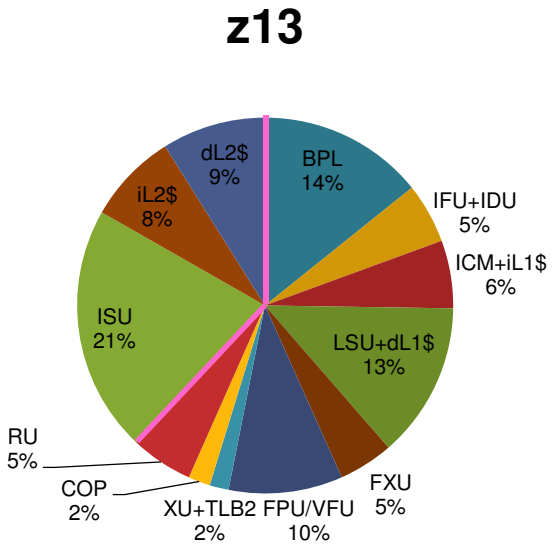
z13 adds SIMD, decimal FP/FX, floating pt divide and VBUs, and doubled existing FXUs and FPUs

Transistor count	60 million	333 million, or 5.55x
Frequency	1.2 GHz	5.0 GHz, or 4.167x

Note the BPL's floor plan percentage doubled on z13 while all other common units (roughly) halved relative to the z990







Why are IBM's processors @ 5+ GHz when Intel's are under 4 GHz?

- System considerations
  - Power delivery to the box
  - Cooling the box
  - IBM's processors are designed for specific IBM systems, while x86 processors target a wide variety of systems; *or IBM sells **systems** – Intel sells **chips***
- Testing / reliability
  - zProcessors all undergo "burn in" and rigorous testing to ensure reliability at higher frequencies
    - Not practical with Intel's massive volumes
- Cost / customer value
  - Pricing model
- All of that to say… the frequency target reflects each business' overall "sweet spot"
- There's also a difference between **nominal** frequency and **maximum** frequency
  - Gamers have overclocked x86's for decades, and motherboard manufacturers readily support this community
  - Some late model under-4 GHz x86's have been liquid-nitrogen cooled and clocked up around 8..9 GHz!
  - No one has ever overclocked a System z to my knowledge…

Flow of this presentation

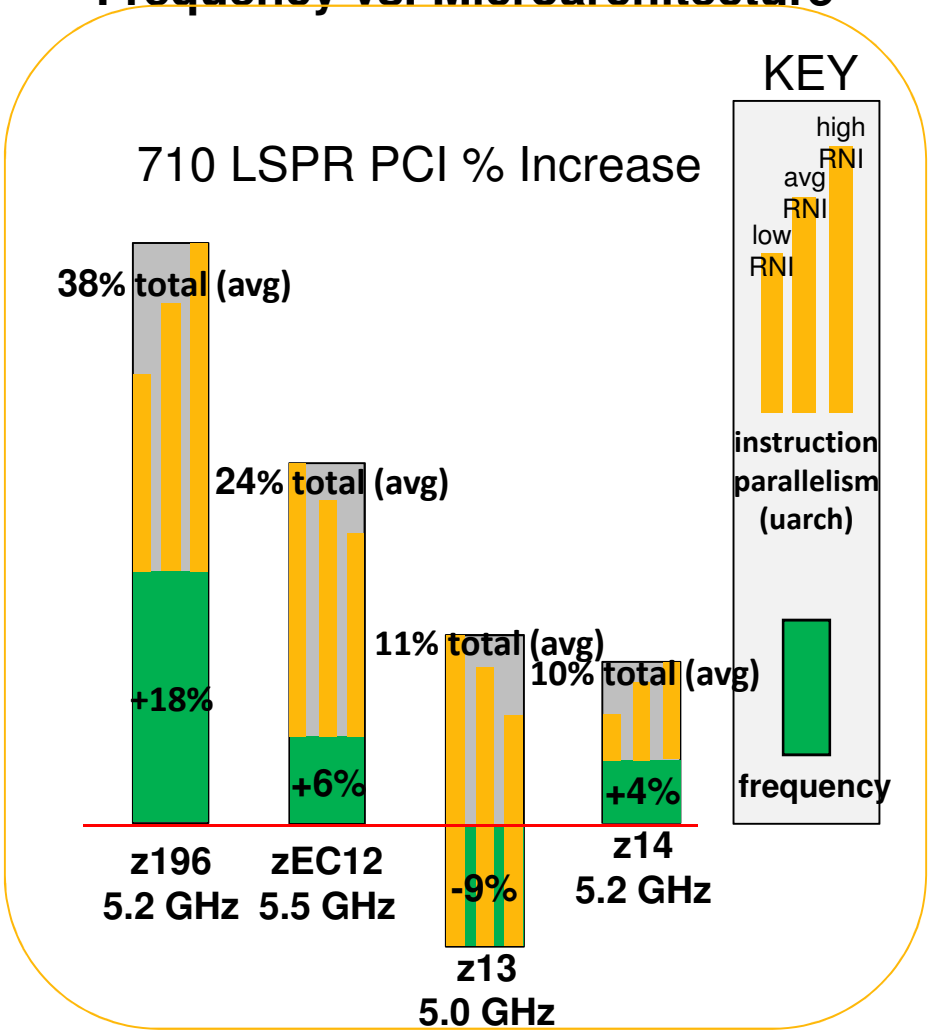
- Section 1: What is a microprocessor? Transistor? VHDL?
- Section 2: Transistor technology
- Section 3: How technology influences logic
- **Section 4: Two design option discussions illustrating the increased importance of well-written software**
  - **L2 I-and-D-cache balancing: deoptimize the store-into-or-near-the-instruction stream case**
  - **Store forwarding improvements: deoptimize the rapid, successive store to the same DW case**
- Section 5: Summary

z13 demarks a new era in IBM Z computing

- Moore's "Law" & Dennard Scaling realities remain challenging
  - Substantial generational frequency gains are history
  - HW/SW synergy is an absolute requirement going forward – Holistic Performance

- z13's design implications
  - Required a hard shift toward microarchitectural improvements
  - As performance gains due to frequency are reduced and those due to microarchitectural features increase, overall performance variability also increases
  - Frequency is a "tide that lifts all boats"

Generational Performance Gains from Frequency vs. Microarchitecture



Design consideration: unified vs separate instruction and data caches

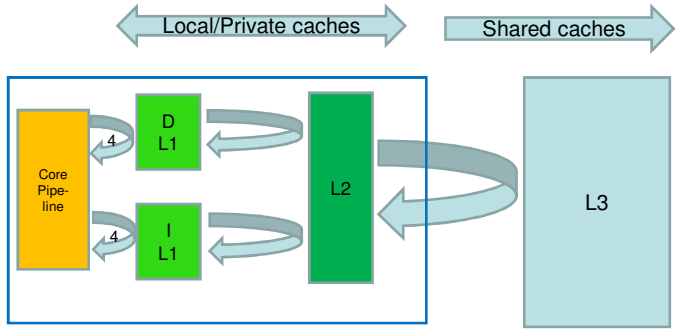
- Allows an instruction cache to be placed near the front end of the pipe and a separate operand-data cache to be placed near the execution units
  - Liberates the floor plan allowing greater placement optimization
  - Allows shorter wires (lower transmission delays and leakage)
- Higher frequency designs are possible as less wall clock time is required to index two smaller caches vs one large one
- Another circuit consideration is the cache array's number of "read ports"
  - An L1 cache array may only service a small number of requests in a cycle – typically two
  - So if both the front end and execution units petition a common, unified cache, the instruction fetches may have to queue up behind the data fetches
- Any drawbacks? Yes – one in particular…
  - It's possible that the same cache line may present in both the instruction and data caches
  - When this occurs the architecture requires both copies must be kept consistent
  - If a store occurs to an active insn cache line, the **synchronization action (cache line copy) is expensive**
    - Of course unified caches have this problem as well, but the synchronization action is relatively cheap (pipe flush)

Why SW optimization matters

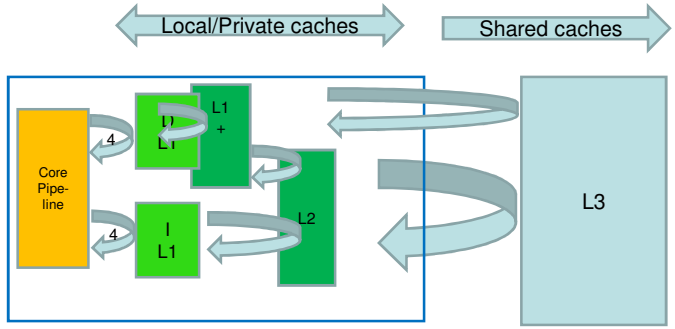
Cache design

- Private (per-core) cache evolution
  - Allows improvements in size and latency
  - Unified vs. split L2 for instructions and operands
    - Split L2 keeps data closer to L1
    - Unified (z196) to hybrid (zEC12) to split (z13/zNext)
  - Integrated vs. serial directory lookup
    - **Integrated reduces access latency for L2, L3**
    - Added for operands (zEC12), Instructions (z13/zNext)
- Allows large, fast L2 caches
  - L2 sizes comparable to others' L3s (MBs)
    - Leverages eDRAM technology
  - Around 10 cycles to access data from L2
- On-chip shared L3
  - Shared by all cores on the CP chip
  - Now also the sharing point for I-L2 and D-L2
- Cache line size is 256B throughout hierarchy
  - Safe value to use for separation / alignment

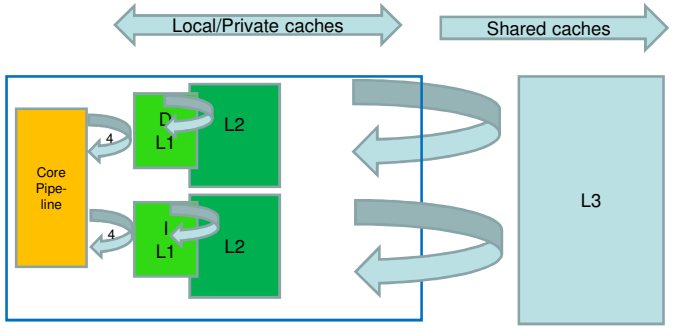
z196 On-chip Cache Hierarchy



zEC12 On-chip Cache Hierarchy



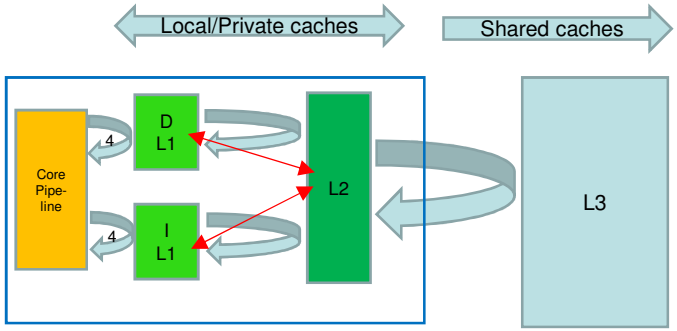
z13-and-z14 On-chip Cache Hierarchy



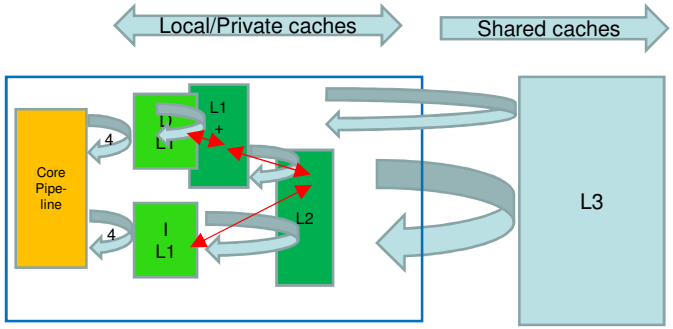
Why SW optimization matters
Instruction / data proximity

- Instructions & Operands in same cache line
  - OK (maybe inefficient) if operands read-only
  - Problem if stores to those operand locations
    - Extra cache misses, long delays
- Split L1 caches (re)introduced in z900 (2000)
  - Designs optimized for well-behaved code
    - Increasing cost of I/D cache contention
  - With split of L2 cache, resolution moved to L3
- Not a problem for
  - Re-entrant code
  - Any LE-based compiler generated code
  - Dynamic run-time code
- Problematic Examples
  - True self-modifying code
  - Classic save area
  - Local save of return address
  - In-line macro parameters
  - Local working area right after code
- Moral of the story: ***don't store near your cached insns in handwritten assembler!***

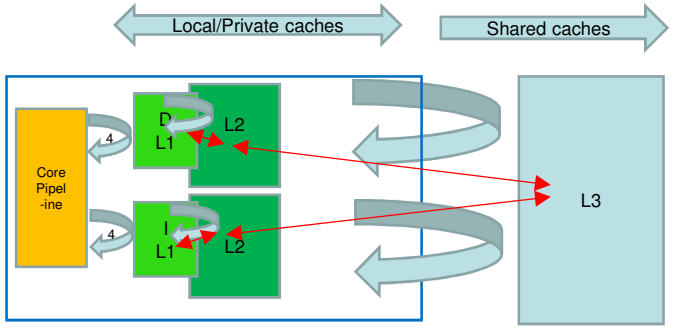
z196 On-chip Cache Hierarchy



zEC12 On-chip Cache Hierarchy



z13-and-z14 On-chip Cache Hierarchy

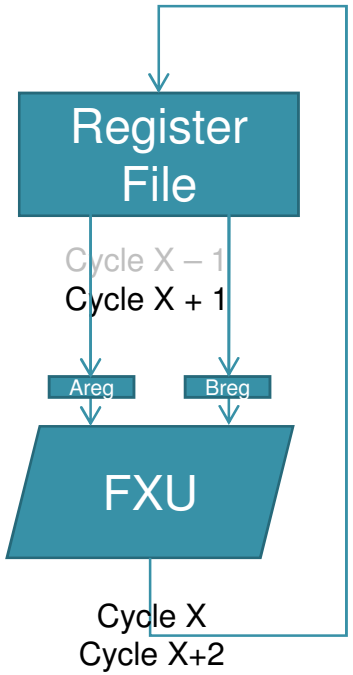


Register bypass paths (background for store data bypass discussion)

Cycle X: AR GR5, GR7

Cycle X + 1: *RF access ("bubble")*

Cycle X + 2: AR GR8, GR5



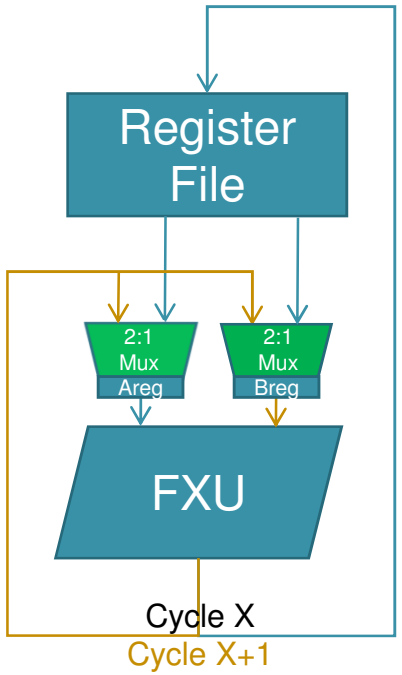
Rectangles ("Register File", "Areg" and "Breg") are latches

Register bypass paths (background for store data bypass discussion)

Cycle X: AR GR5, GR7

~~Cycle X + 1: *RF access ("bubble")*~~

Cycle X + 1: AR GR8, GR5



Rectangles ("Register File", "Areg" and "Breg") are latches

*By double-dropping the FXU's result bus to both the register file *and* back into the FXU's own A-and-B-regs as a "hot bypass", the bubble cycle between back-to-back-and-execution-dependent FXU operations can be eliminated.*

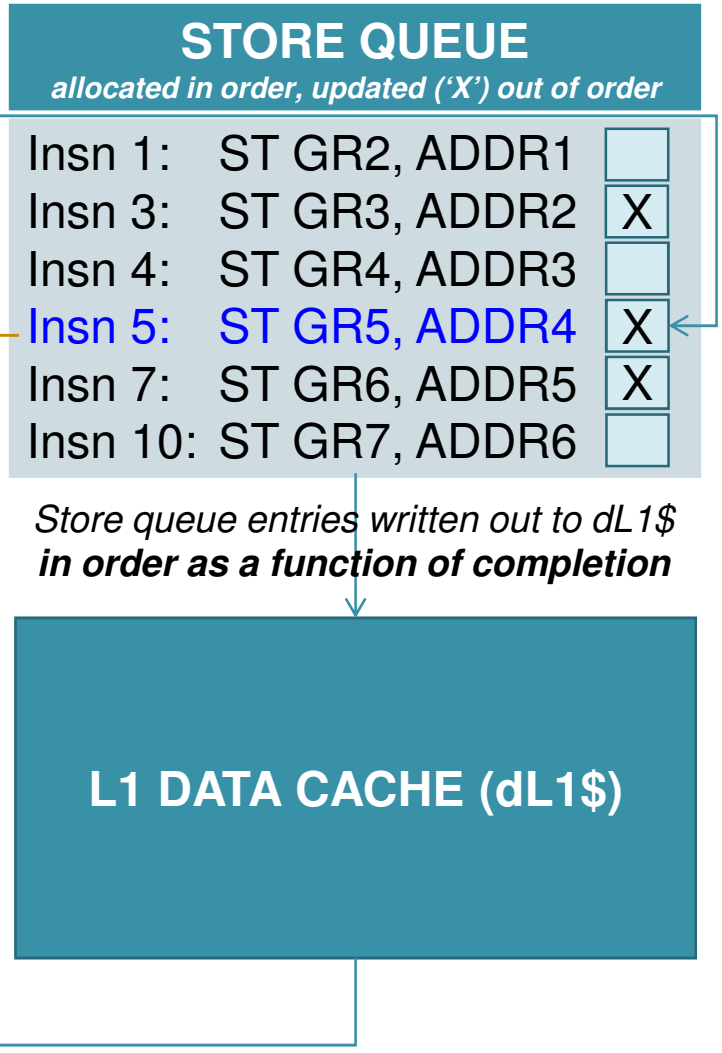
Storage data bypass paths

```
Insn 5:   ST GR5, ADDR4
Insn 6:   AR GR5, GR6
Insn 7:   ST GR6, ADDR5
Insn 8:   AR GR7, GR5
Insn 9:   L GR1, ADDR4
```

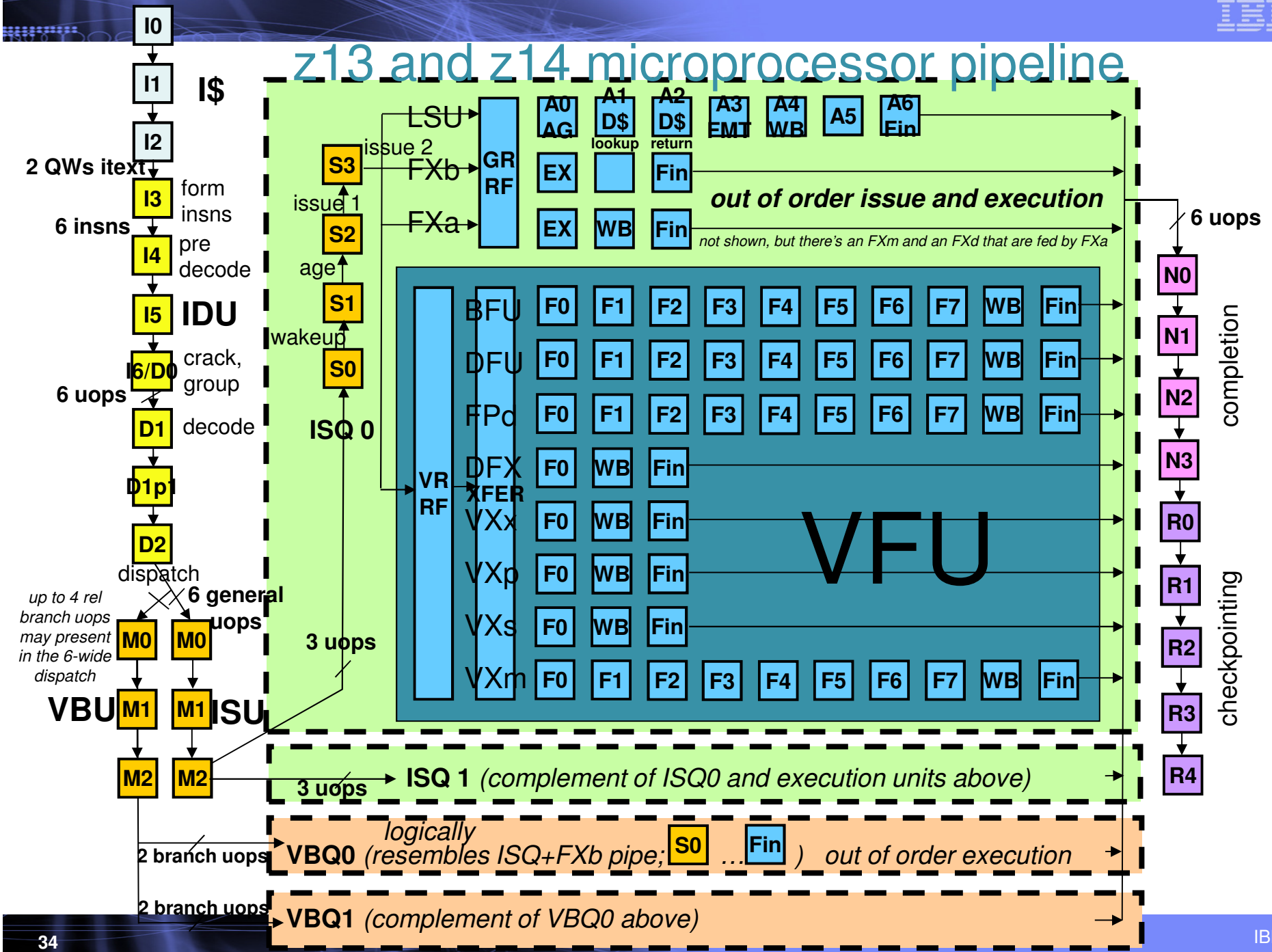
Store forwarding: bypassing Insn 5's store data out of the store queue (or a parallel structure) **at execution time** allows Insn 9 to execute **very quickly** after Insn 5.

Without store forwarding: data from the store queue must be **written out to the dL1** in order **at a function of completion time**, which may require Insn 9 wait a **long time** to receive Insn 5's stored data.

Best case z13 dL1 write is N14 / R10
Best case z14 dL1 write is N15 / R11

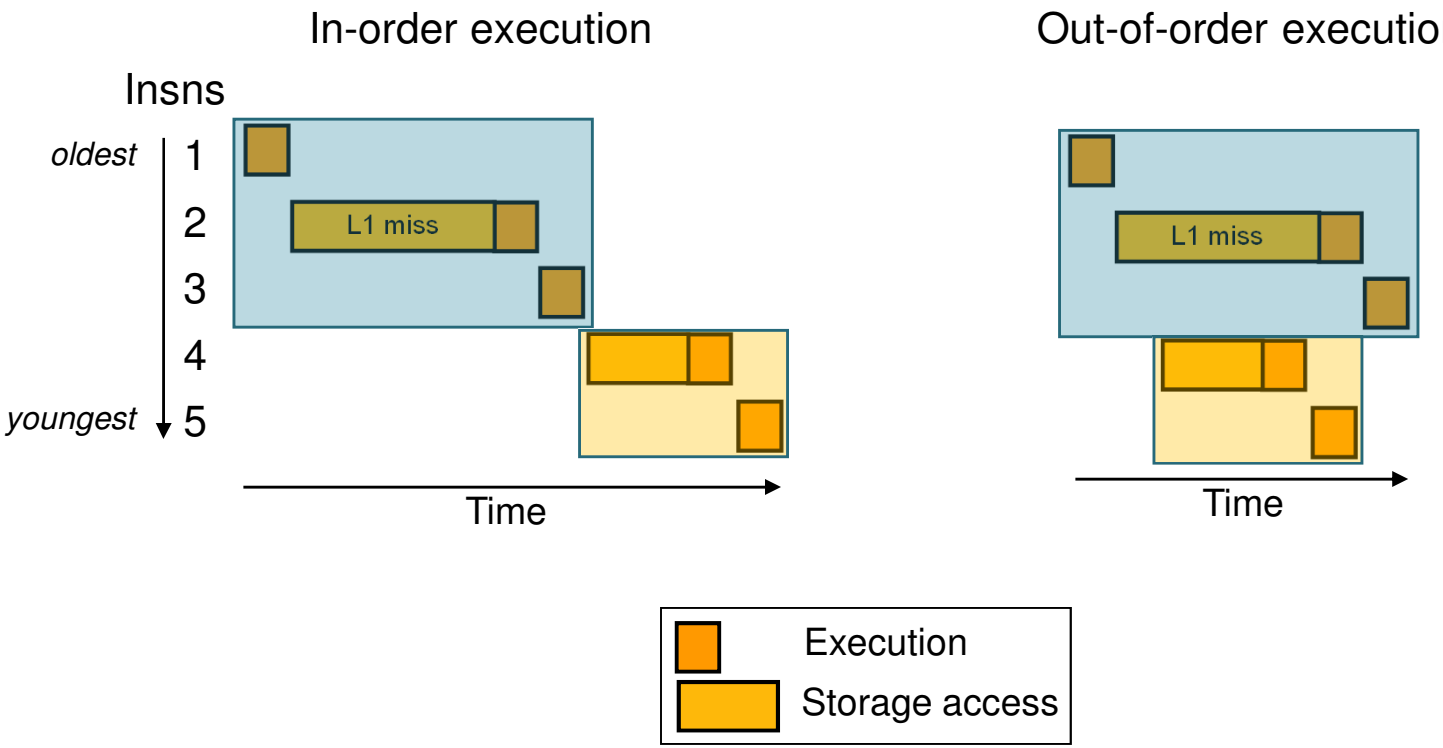


z13 and z14 microprocessor pipeline



Out-of-order execution

- Insn 2 is AGEN-dependent on insn 1
- Insn 3 is execution-dependent on insn 2 <= insns in blue box denote dependency chain ‘1’
- Insn 5 is execution-dependent on insn 4 <= insns in tan box denote dependency chain ‘2’



- ***NOTE: not shown, but in-order completion is an architectural requirement!***

Microprocessor hardware comparison:
zEC12's store queue (STQ) w/ store forwarding buffer (SFB) vs. z13 and z14's STQ plus 4-set-associative store forwarding cache (SFC)

Feature	zEC12	z13 & z14
Can stitch together the data to be forwarded from multiple SFB/SFC entries and/or the L1 cache	Yes	Yes
Footprint	16 DWs	128 DWs
Bytes forwardable (potentially)	Final DW	All
SMT capable	No	Yes
Susceptible to younger databeats executing OOO and preventing bypassing to older loads by even older databeats	No	Yes
4-set-associative design allows at most 4 in-order databeats to the same 56:60 before overwriting an older entry	No	Yes

- Moral of the story: ***don't do a flurry of stores to the same DW!***
  - Example: COBOL using a suboptimal USAGE IS DISPLAY type as an array index or a loop iterator

Flow of this presentation

- Section 1: What is a microprocessor? Transistor? VHDL?
- Section 2: Transistor technology
- Section 3: How technology influences logic
- Section 4: Two design option discussions illustrating the increased importance of well-written software
  - L2 I-and-D-cache balancing: deoptimize the store-into-or-near-the-instruction stream case
  - Store forwarding improvements: deoptimize the rapid, successive store to the same DW case
- **Section 5: Summary**

HW summary

- The 5.0 GHz z13 and 5.2 GHz z14 processors generally outperform their 5.5 GHz zEC12 predecessor by ~10% and ~20% respectively (LSPR) *despite their clock frequencies being ~9% and ~5.5% slower*
  - **z13 and z14's higher-density technology and lower frequency (and proportionally lower leakage current) design affords numerous perks**
    - Double the decode bandwidth of zEC12 plus more execution pipes
    - An SMT2 design that collaterally increases single-threaded performance
    - Larger L1 and L2 on-core caches, larger L3 on-CP-chip cache
  - **z13 and z14 optimized the L1 <-> L2 insn cache interface**
    - The tradeoff deoptimized the L1-data-cache-to-L1-insn-cache synchronization logic
      - This makes poorly-handwritten assembler run worse on the z13 than the zEC12
    - However it makes well-written code perform better as the L2 instruction cache latency is reduced by 33% (12 cycle access down to 8 cycles)
  - **z13 and z14's intracore store data bypass design is generally superior to the zEC12's**
    - It doesn't stomach multiple, successive stores to the same location as well as zEC12
    - However it 1) covers 8x the data, 2) permits bypassing from any part of a storing insn (not just the tail end), and 3) facilitates an SMT2 design that collectively nets a win in most all cases

SW summary

- HW/SW synergy is essential in this post-generational-frequency-bump era of computing
  - **The latest compilers use the latest instruction set architecture (ISA)**
    - RISBG/NIHF/OIHF example (and there are countless others) – *from **storage-based** variable manipulation to purely **register-based**:*

```
ST    12,FLOATJSR          STORE SIGN BIT
NI    FLOATJSR,X'80'       RETAIN SIGN BIT
OI    FLOATJSR,X'4E'       EXPONENT
XC    FLOATJSR+1(3),FLOATJSR+1
LPR   12,12                GENERAL REGISTER MADE POSITIVE
ST    12,FLOATJSR+4
AD    0,FLOATJSR           NORMALIZED FLOATING POINT NUMBER
```

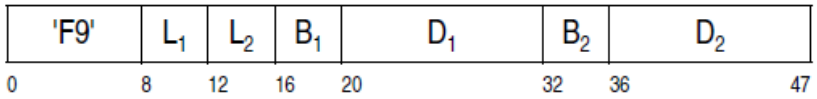
```
RISBG 12,12,0,0,32     COPY SIGN BIT
NIHF  12,X'80000000'   RETAIN SIGN BIT
OIHF  12,X'4E000000'   EXPONENT
LPR   12,12            GENERAL REGISTER MADE POSITIVE
LDGR  12,12            GR12 -> FPR12
ADR   0,12             NORMALIZED FLOATING POINT NUMBER
```

    - **SIMD for packed decimal (COBOL) operations available on z14**
  - **The latest compilers tune their scheduling to the latest processor's specific microarchitecture**
    - If decode and dispatch are 3-uops wide, the compiler will attempt to schedule 3-uop groups
    - If multiple instruction combination solutions exist to perform a specific action, the compiler will choose the optimal sequence for a given processor
    - Modern compilers presume split instruction and data caches and proactively separate instructions from data
  - If you have handwritten assembler, the onus to remain current is on you, *as it has always been*
  - ***The consequences of not remaining current are much higher now than they were in the past***

New to z14: Vector Register-based Decimal Architecture

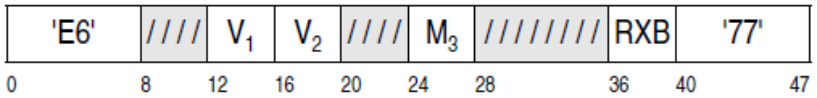
COMPARE DECIMAL

CP $D_1(L_1,B_1),D_2(L_2,B_2)$ [SS-b]



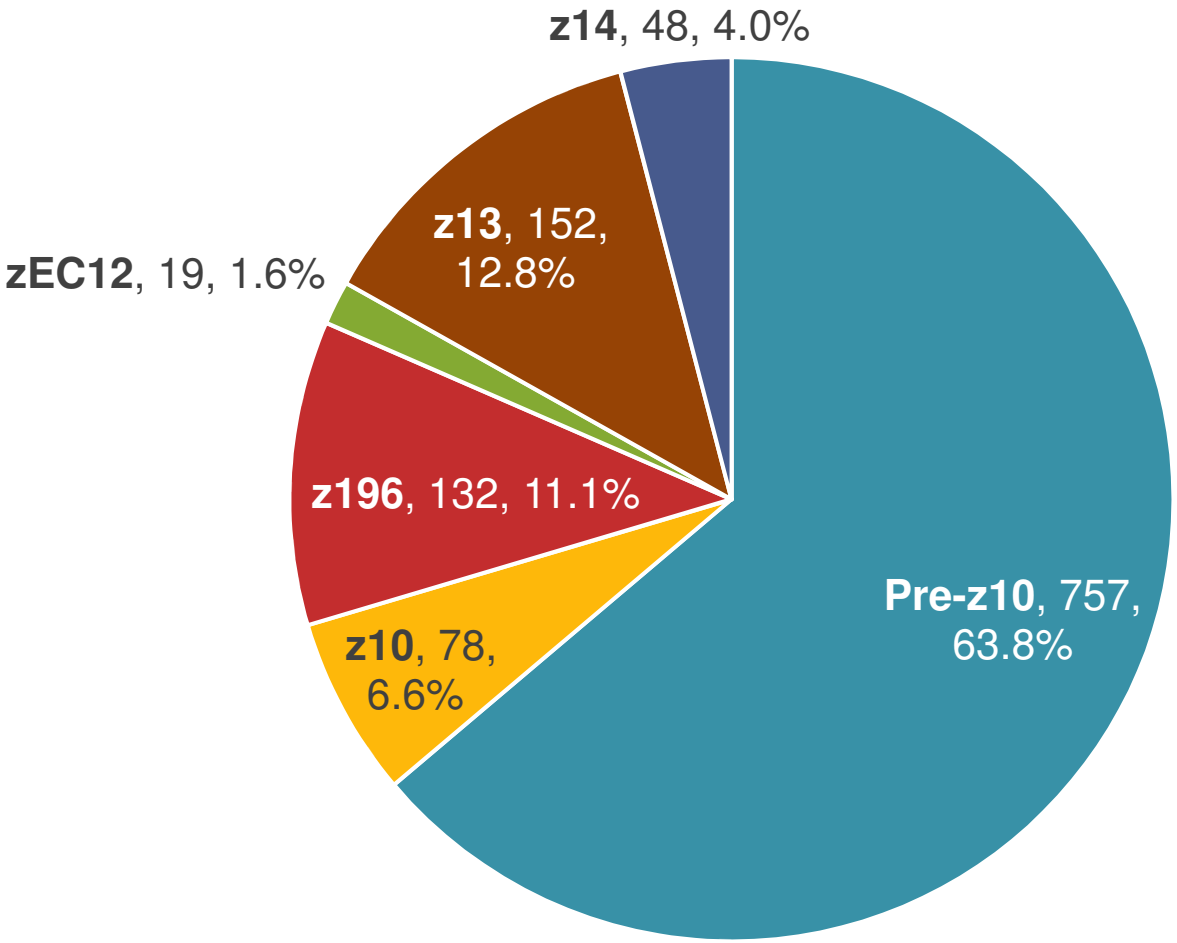
VECTOR COMPARE DECIMAL

VCP V_1,V_2,M_3 [VRR-h]



Old Decimal Insn	New Vector Insn
CP	VCP
AP	VAP
SP	VSP
MP	VMP
DP	VDP
SRP	VSRP
PACK	VPKZ
UNPK	VUPKZ
CVD/CVD8	VCD/VCVDG
CVB/CVB8	VCB/VCVBG
COMPP/ABSP/DECFIX	VPSOP
(N/A)	VLRL/VSTRL
ZAP/MVC	(N/A)

IBM Z Instructions per Generation, 1186 total



IBM Z Instructions per Generation, 1186 total

- **Pre-z10 (757 or 64%):** A, AD, ADB, ADBR, ADR, ADTR, AE, AEB, AEBR, AER, AFI, AG, AGF, AGFI, AGFR, AGHI, AGR, AH, AHI, AHY, AL, ALC, ALCG, ALCGR, ALCR, ALFI, ALG, ALGF, ALGFI, ALGFR, ALGR, ALR, ALY, AP, AR, AU, AUR, AW, AWR, AXBR, AXR, AXTR, AY, BAKR, BAL, BALR, BAS, BASR, BASSM, BC, BCR, BCT, BCTG, BCTGR, BCTR, BRAS, BRASL, BRC, BRCL, BRCT, BRCTG, BRXH, BRXHG, BRXLE, BRXLG, BSA, BSG, BSM, BXH, BXHG, BXLE, BXLEG, C, CD, CDB, CDBR, CDFBR, CDFR, CDGBR, CDGR, CDGTR, CDR, CDS, CDSG, CDSTR, CDSY, CDTR, CDUTR, CE, CEB, CEBR, CEDTR, CEFBR, CEFR, CEGBR, CEGR, CER, CEXTR, CFC, CFDBR, CFDR, CFEBR, CFER, CFI, CFXBR, CFXR, CG, CGDBR, CGDR, CGDTR, CGEBR, CGER, CGF, CGFI, CGFR, CGHI, CGR, CGXBR, CGXR, CGXTR, CH, CHI, CHY, CKSM, CL, CLC, CLCL, CLCLE, CLCLU, CLFI, CLG, CLGF, CLGFI, CLGFR, CLGR, CLI, CLIY, CLM, CLMH, CLMY, CLR, CLST, CLY, CMPSC, CP, CPSDR, CPYA, CR, CS, CSCH, CSDTR, CSG, CSP, CSPG, CSST, CSXTR, CSY, CU12, CU14, CU21, CU24, CU41, CU42, CUDTR, CUSE, CUTFU, CUUTF, CUXTR, CVB, CVBG, CVBY, CVD, CVDG, CVDY, CXBR, CXFBR, CXFR, CXGBR, CXGR, CXGTR, CXR, CXSTR, CXTR, CXUTR, CY, D, DD, DDB, DDBR, DDR, DDTR, DE, DEB, DEBR, DER, DIDBR, DIEBR, DL, DLG, DLGR, DLR, DP, DR, DSG, DSGF, DSGFR, DSGR, DXBR, DXR, DXTR, EAR, ECTG, ED, EDMK, EEDTR, EEXTR, EFPC, EPAIR, EPAR, EPSW, EREG, EREGG, ESAIR, ESAR, ESDTR, ESEA, ESTA, ESXTR, EX, FIDBR, FIDR, FIDTR, FIEBR, FIER, FIXBR, FIXR, FIXTR, FLOGR, HDR, HER, HSCH, IAC, IC, ICM, ICMH, ICMY, ICY, IDTE, IEDTR, IEXTR, IIHF, IIHH, IIHL, IILF, IILH, IILL, IPK, IPM, IPTE, ISKE, IVSK, KDB, KDBR, KDTR, KEB, KEBR, KIMD, KLMD, KM, KMAC, KMC, KXBR, KXTR, L, LA, LAE, LAM, LAMY, LARL, LASP, LAY, LB, LBR, LCDBR, LCDFR, LCDR, LCEBR, LCER, LCGFR, LCGR, LCR, LCTL, LCTLG, LCXBR, LCXR, LD, LDE, LDEB, LDEBR, LDER, LDETR, LDGR, LDR, LDXBR, LDXR, LDXTR, LDY, LE, LEDBR, LEDR, LEDTR, LER, LEXBR, LEXR, LEY, LFAS, LFPC, LG, LGB, LGBR, LGDR, LGF, LGFI, LGFR, LGH, LGHI, LGHR, LGR, LH, LHI, LHR, LHY, LLC, LLCR, LLGC, LLGCR, LLGF, LLGFR, LLGH, LLGHR, LLGT, LLGTR, LLH, LLHR, LLIHF, LLIHH, LLIHL, LLILF, LLILH, LLILL, LM, LMD, LMG, LMH, LMY, LNDBR, LNDFR, LNDR, LNEBR, LNER, LNGFR, LNGR, LNR, LNXBR, LNXR, LPDBR, LPDFR, LPDR, LPEBR, LPER, LPGFR, LPGR, LPQ, LPR, LPSW, LPSWE, LPTEA, LPXBR, LPXR, LR, LRA, LRAG, LRAY, LRDR, LRER, LRV, LRVG, LRVGR, LRVH, LRVR, LT, LTDBR, LTDR, LTDTR, LTEBR, LTER, LTG, LTGFR, LTGR, LTR, LTXBR, LTXR, LTXTR, LURA, LURAG, LXD, LXDB, LXDBR, LXDR, LXDTR, LXE, LXEB, LXEBR, LXER, LXR, LY, LZDR, LZER, LZXR, M, MAD, MADB, MADBR, MADR, MAE, MAEB, MAEBR, MAER, MAY, MAYH, MAYHR, MAYL, MAYLR, MAYR, MC, MD, MDB, MDBR, MDE, MDEB, MDEBR, MDER, MDR, MDTR, ME, MEE, MEEB, MEEBR, MEER, MER, MGHI, MH, MHI, ML, MLG, MLGR, MLR, MP, MR, MS, MSCH, MSD, MSDB, MSDBR, MSDR, MSE, MSEB, MSEBR, MSER, MSG, MSGF, MSGFR, MSGR, MSR, MSTA, MSY, MVC, MVCDK, MVCIN, MVCK, MVCL, MVCLE, MVCLU, MVCOS, MVCP, MVCS, MVCSK, MVI, MVIY, MVN, MVO, MVPG, MVST, MVZ, MXBR, MXD, MXDB, MXDBR, MXDR, MXR, MXTR, MY, MYH, MYHR, MYL, MYLR, MYR, N, NC, NG, NGR, NI, NIHF, NIHH, NIHL, NILF, NILH, NILL, NIY, NR, NY, O, OC, OG, OGR, OI, OIHF, OIHH, OIHL, OILF, OILH, OILL, OIY, OR, OY, PACK, PALB, PC, PFPO, PGIN, PGOUT, PKA, PKU, PLO, PR, PT, PTFF, PTI, PTLB, QADTR, QAXTR, RCHP, RLL, RLLG, RP, RRBE, RRDTR, RRXTR, RSCH, S, SAC, SACF, SAL, SAM24, SAM31, SAM64, SAR, SCHM, SCK, SCKC, SCKPF, SD, SDB, SDBR, SDR, SDTR, SE, SEB, SEBR, SER, SFASR, SFPC, SG, SGF, SGFR, SGR, SH, SHY, SIGP, SL, SLA, SLAG, SLB, SLBG, SLBGR, SLBR, SLDA, SLDL, SLDT, SLFI, SLG, SLGF, SLGFI, SLGFR, SLGR, SLL, SLLG, SLR, SLXT, SLY, SP, SPKA, SPM, SPT, SPX, SQD, SQDB, SQDBR, SQDR, SQE, SQEB, SQEBR, SQER, SQXBR, SQXR, SR, SRA, SRAG, SRDA, SRDL, SRDT, SRL, SRLG, SRNM, SRNMT, SRP, SRST, SRSTU, SRXT, SSAIR, SSAR, SSCH, SSKE, SSM, ST, STAM, STAMY, STAP, STC, STCK, STCKC, STCKE, STCKF, STCM, STCMH, STCMY, STCPS, STCRW, STCTG, STCTL, STCY, STD, STDY, STE, STEY, STFL, STFLE, STFPC, STG, STH, STHY, STIDP, STM, STMG, STMH, STMY, STNSM, STOSM, STPQ, STPT, STPX, STRAG, STRV, STRVG, STRVH, STSCH, STSI, STURA, STURG, STY, SU, SUR, SVC, SW, SWR, SXBR, SXR, SXTR, SY, TAM, TAR, TB, TBDR, TBEDR, TCDB, TCEB, TCXB, TDCDT, TDCET, TDCXT, TDGDT, TDGET, TDGXT, THDER, THDR, TM, TMH, TMHH, TMHL, TML, TMLH, TMLL, TMY, TP, TPI, TPROT, TR, TRACE, TRACG, TRAP2, TRAP4, TRE, TROO, TROT, TRT, TRTO, TRTR, TRTT, TS, TSCH, UNPK, UNPKA, UNPKU, UPT, X, XC, XG, XGR, XI, XIHF, XILF, XIY, XR, XSCH, XY, ZAP
- **z10 (78 or 6.5%):** AGSI, ALGSI, ALSI, ASI, CGFRL, CGH, CGHRL, CGHSI, CGIB, CGIJ, CGIT, CGRB, CGRJ, CGRL, CGRT, CHHSI, CHRL, CHSI, CIB, CIJ, CIT, CLFHSI, CLFIT, CLGFRL, CLGHRL, CLGHSI, CLGIB, CLGIJ, CLGIT, CLGRB, CLGRJ, CLGRL, CLGRT, CLHHSI, CLHRL, CLIB, CLIJ, CLRB, CLRJ, CLRL, CLRT, CRB, CRJ, CRL, CRT, ECAG, EXRL, LAEY, LGFRL, LGHRL, LGRL, LHRL, LLGFRL, LLGHRL, LLHRL, LRL, LTGF, MFY, MHY, MSFI, MSGFI, MVGHI, MVHHI, MVHI, PCKMO, PFD, PFDRL, PFMF, PTF, RISBG, RNSBG, ROSBG, RXSBG, STGRL, STHRL, STRL, TRTE, TRTRE
- **z196 (132 or 11%):** ADTRA, AGHIK, AGRK, AHHHR, AHHLR, AHIK, AIH, ALGHSIK, ALGRK, ALHHHR, ALHHLR, ALHSIK, ALRK, ALSIH, ALSIHN, ARK, AXTRA, BRCTH, CDFBRA, CDFTR, CDGBRA, CDGTRA, CDLFBR, CDLFTR, CDLGBR, CDLGTR, CEFBRA, CEGBRA, CELFBR, CELGBR, CFDBRA, CFDTR, CFEBRA, CFXBRA, CFXTR, CGDBRA, CGDTRA, CGEBRA, CGXBRA, CGXTRA, CHF, CHHR, CHLR, CIH, CLFDBR, CLFDTR, CLFEBR, CLFXBR, CLFXTR, CLGDBR, CLGDTR, CLGEBR, CLGXBR, CLGXTR, CLHF, CLHHR, CLHLR, CLIH, CXFBRA, CXFTR, CXGBRA, CXGTRA, CXLFBR, CXLFTR, CXLGBR, CXLGTR, DDTRA, DXTRA, FIDBRA, FIEBRA, FIXBRA, KMCTR, KMF, KMO, LAA, LAAG, LAAL, LAALG, LAN, LANG, LAO, LAOG, LAX, LAXG, LBH, LDXBRA, LEDBRA, LEXBRA, LFH, LHH, LLCH, LLHH, LOC, LOCG, LOCGR, LOCR, LPD, LPDG, MDTRA, MXTRA, NGRK, NRK, OGRK, ORK, PCC, PLM, POPCNT, RISBHG, RISBLG, RRBM, SDTRA, SGRK, SHHHR, SHHLR, SLAK, SLGRK, SLHHHR, SLHHLR, SLLK, SLRK, SRAK, SRK, SRLK, SRNMB, STCH, STFH, STHH, STOC, STOCG, SXTRA, XGRK, XRK
- **zEC12 (19 or 1.5%):** BPP, BPRP, CDZT, CLGT, CLT, CRDTE, CXZT, CZDT, CZXT, ETND, NIAI, NTSTG, PPA, PRNO, RISBGN, TABORT, TBEGIN, TBEGINC, TEND
- **z13 (152 or 13%):** CDPT, CPDT, CPXT, CXPT, LCCB, LLZRGF, LOCFH, LOCFHR, LOCGHI, LOCHHI, LOCHI, LZRF, LZRG, STOCFH, VA, VAC, VACC, VACCC, VAVG, VAVGL, VCDG, VCDLG, VCEQ, VCGD, VCH, VCHL, VCKSM, VCLGD, VCLZ, VCTZ, VEC, VECL, VERIM, VERLL, VERLLV, VESL, VESLV, VESRA, VESRAV, VESRL, VESRLV, VFA, VFAE, VFCE, VFCH, VFCHE, VFD, VFEE, VFENE, VFI, VFLL, VFLR, VFM, VFMA, VFMS, VFPSO, VFS, VFSQ, VFTCI, VGBM, VGEF, VGEG, VGFM, VGFMA, VGM, VISTR, VL, VLBB, VLC, VLEB, VLEF, VLEG, VLEH, VLEIB, VLEIF, VLEIG, VLEIH, VLGV, VLL, VLLEZ, VLM, VLP, VLR, VLREP, VLVG, VLVGP, VMAE, VMAH, VMAL, VMALE, VMALH, VMALO, VMAO, VME, VMH, VML, VMLE, VMLH, VMLO, VMN, VMNL, VMO, VMRH, VMRL, VMX, VMXL, VN, VNC, VNO, VO, VPDI, VPERM, VPK, VPKLS, VPKS, VPOPCT, VREP, VREPI, VS, VSBCBI, VSBI, VSCBI, VSCEF, VSCEG, VSEG, VSEL, VSL, VSLB, VSLDB, VSRA, VSRAB, VSRL, VSRLB, VST, VSTEB, VSTEF, VSTEG, VSTEH, VSTL, VSTM, VSTRC, VSUM, VSUMG, VSUMQ, VTM, VUPH, VUPL, VUPLH, VUPLL, VX, WFC, WFK
- **z14 (48 or 4%):** AGH, BIC, IRBM, KMA, LGG, LGSC, LLGHSG, MG, MGH, MGRK, MSC, MSGC, MSGRKC, MSRKC, SGH, STGSC, TPEI, VAP, VBPRM, VCP, VCVB, VCVBG, VCVD, VCVDG, VDP, VFMAX, VFMIN, VFNMA, VFNMS, VLIP, VLRL, VLRLR, VMP, VMSL, VMSL, VNN, VNX, VOC, VPKZ, VPSOP, VRP, VSDP, VSP, VSRP, VSTRL, VSTRLR, VTP, VUPKZ

What's the net of all of this? A roundabout way of saying…

We'll continue to innovate and push the boundaries of classical physics for the foreseeable future.

In return we ask that you please make an effort to maintain/update your handwritten assembler and keep current on your compilers!

Thanks for coming!

hutton@us.ibm.com

BACKUP

Microprocessor lingo

- "Pipeline"
  - Assembly-line, overlapped instruction processing – typically instruction-text fetch -> decode -> issue -> execute -> record results
- "Superscalar" vs "Single-Scalar"
  - Describes multiple parallel decode, dispatch and execution paths
  - "2-way superscalar" means two uops may decode, dispatch and execute in parallel
  - Can also say "3-way superscalar decode and dispatch, 6-way superscalar execution", etc
- "Out of order" (or "OOO")
  - Typically describes execution but may also describe instruction and operand fetching; *note IO fetching may resolve OOO*
  - Allows younger uops or fetches to issue around delayed older uops or fetches
    - Younger uops cannot be "dependent" on the older uops they issue around
  - OOO execution mitigates latencies associated with data cache misses and general dependency resolution
- "Register renaming"
  - The use of a large pool of registers to hold not just the architected register contents but a sizeable volume of temporary copies for updating by in-flight (and possibly OOO) insns/uops
    - Say there are three in-flight insns that update GR 5, so four entries in the register pool reflect GR5 in its various temporal states: three for the in-flight updates plus the base, architected copy
  - The scheduler manages the register pool and knows which copy to supply as input to downstream insns/uops
- "Architecture" vs "Microarchitecture"
  - The former is the specification (e.g., POPs), the latter is any elective embodiment of the architecture
  - NOTE: the S/360 was the first computer system to decouple its architecture from its microarchitecture

Microprocessor lingo (continued)

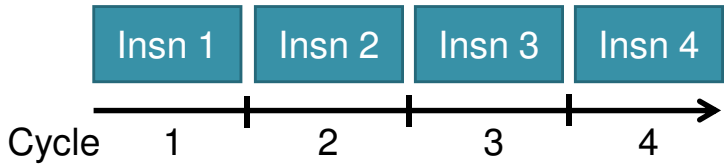
- "CISC" / "RISC" / complex insns / simple insns
  - Complex/Reduced Instruction Set Computing – describes a computer architecture
    - RISC architectures exclusively contain simple insns, CISC architectures are all non-RISC architectures
  - In zArchitecture, simple insns are generally RR-fmt arithmetics and RX-fmt loads and stores
  - Complex insns are everything else: RX-fmt arithmetics, SS-fmt insns (e.g., MVC), POPs Chapter 10 insns, etc.
- "Cracking" or "expansion", and "micro-ops"
  - The processor itself devolving a single (generally-complex) insn into a series of more easily-digestible/executable instruction "micro-operations" (or "uops")
- "Licensed Internal Code" (or "LIC")
  - Proprietary subroutines that reside in secure storage and allow single complex insns to be executed as a series of simpler insns, each of which may be further cracked/expanded into uops at the core's discretion
- "Insn (or uop) Dependencies"
  - If older uop A writes GR 5 and younger uop B reads GR 5, then uop B is dependent upon uop A
  - A uop may be "execution-dependent" or "address generation ("AGEN")-dependent" on a prior uop
- "Simultaneous Multithreading" (or SMT)
  - Describes the processing of multiple user programs concurrently
  - Requires the tracking of an instance of the machine state for each thread
  - Like OOO execution, SMT also mitigates latencies associated with data cache misses and general dependency resolution

General pipelined microprocessor logic design

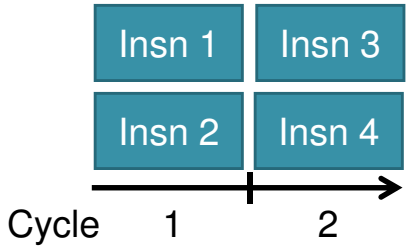
- Dynamic Branch Prediction
  - Conceptually similar to a cache in that it's also primarily a large array indexed by an address, but instead of instructions and operand data its contents are vectors describing branch history near that address
  - It runs as its own pipeline that proactively redirects the main pipeline when it believes a taken-branch is encountered, and it also monitors the downstream resolution of all branches in the main pipeline which it uses to update its history vectors
- Instruction fetching
  - Generally sequential unless redirected by predicted-taken branches
  - Brings instruction lines in from the Level 1 instruction cache to the local instruction text buffers
  - Preliminary instruction parsing (identify instruction boundaries)
- Decode
  - Cracking of complex insns into simpler uop(s) (high frequency design)
  - Determine each uop's dependencies and execution unit
  - Process non-dynamically-predicted relative branches (register-based branch-target address generation not required)
- Dispatch
  - The handoff of decoded insns or uops to the scheduler (if OOO design) or their respective execution unit (if in-order design)
- Instruction/uop scheduling plus issue (out-of-order design)
  - Monitor the dependency-resolution status of all uops and issue ready uop(s) to their respective execution unit(s)
- Execution
  - Fetch operand data from the data cache or store bypass buffer, receive register data from the regfiles or bypass paths, etc
  - Do the work; e.g., produce the updated machine state for each micro op
  - Process non-dynamically-predicted non-relative branches, and resolve all branches
- Record results – Informally record new machine state (update in flight register copies), though we may still take an exception
- Completion
  - Once we know there are no exceptions or flags to be raised, formally record new machine state (update architected register copies)
  - Store updated data out to the L1 data cache

Instruction pipelining – completion view

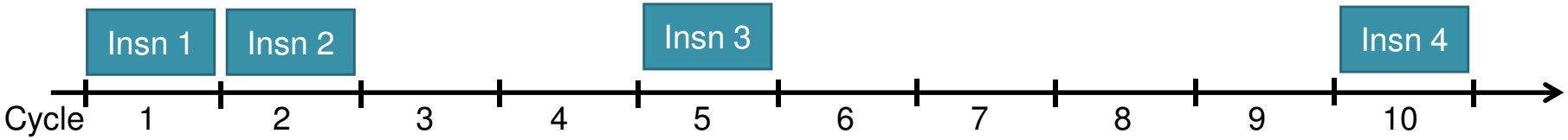
Single-scalar completion of simple insns



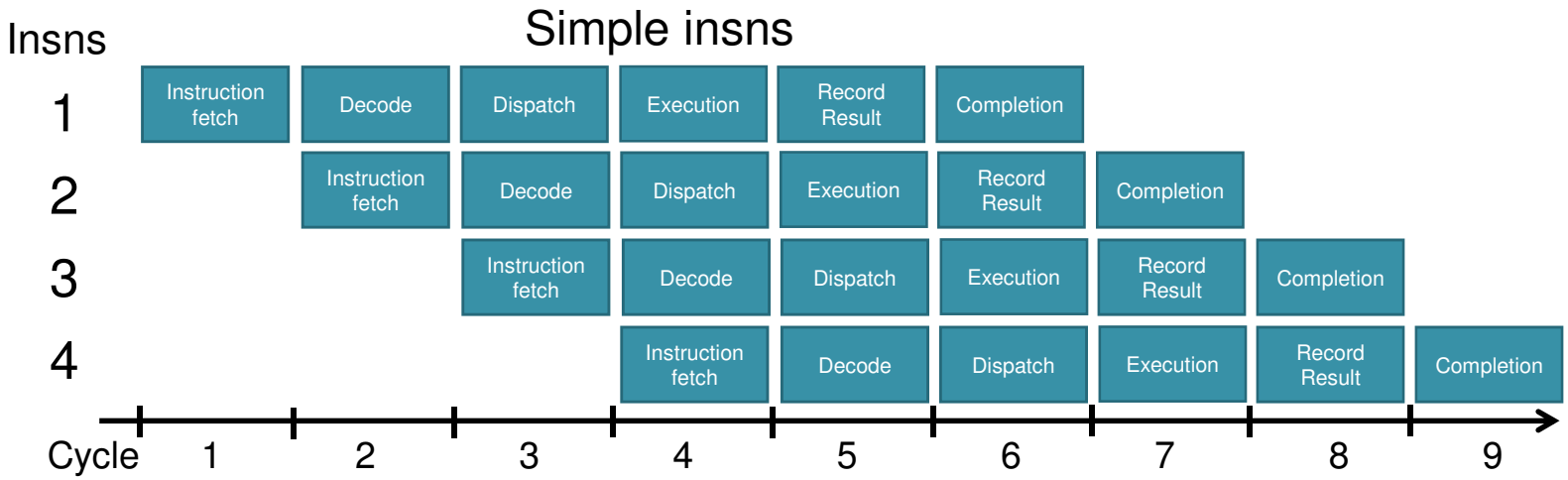
2-way superscalar completion of simple insns

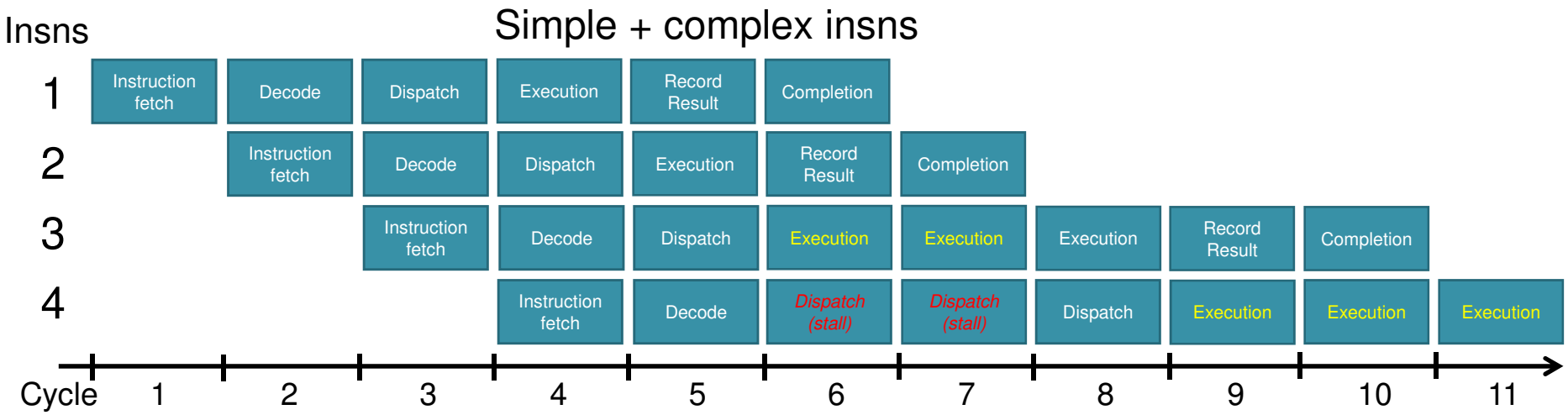


Single-scalar completion of simple (insns 1 and 2) and complex insns (3 and 4)



Instruction pipelining – single-scalar, in-order full-pipe view





Instruction pipelining – 2-way superscalar, in-order full-pipe view