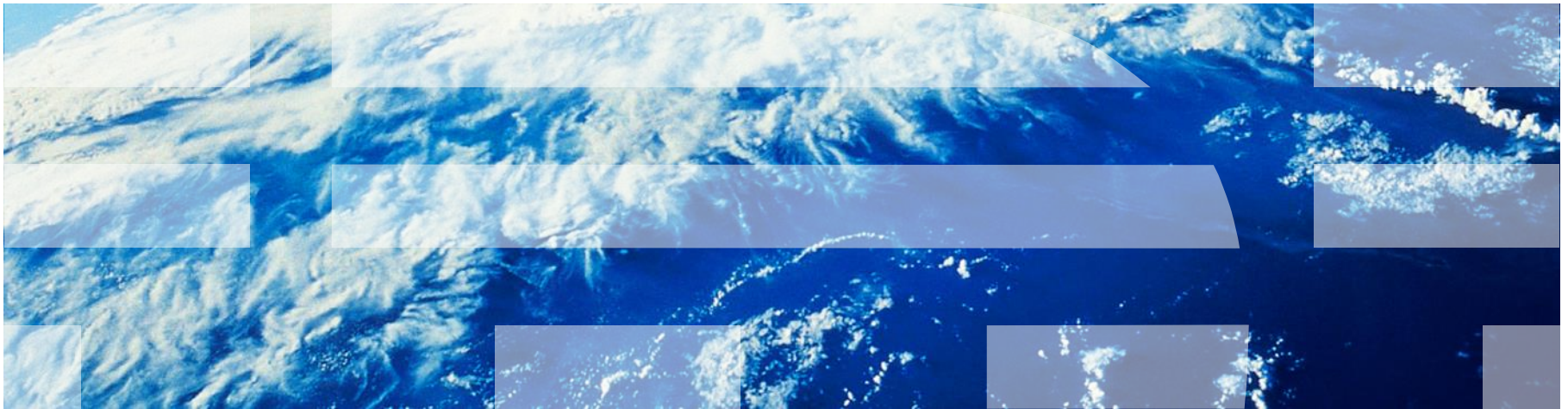
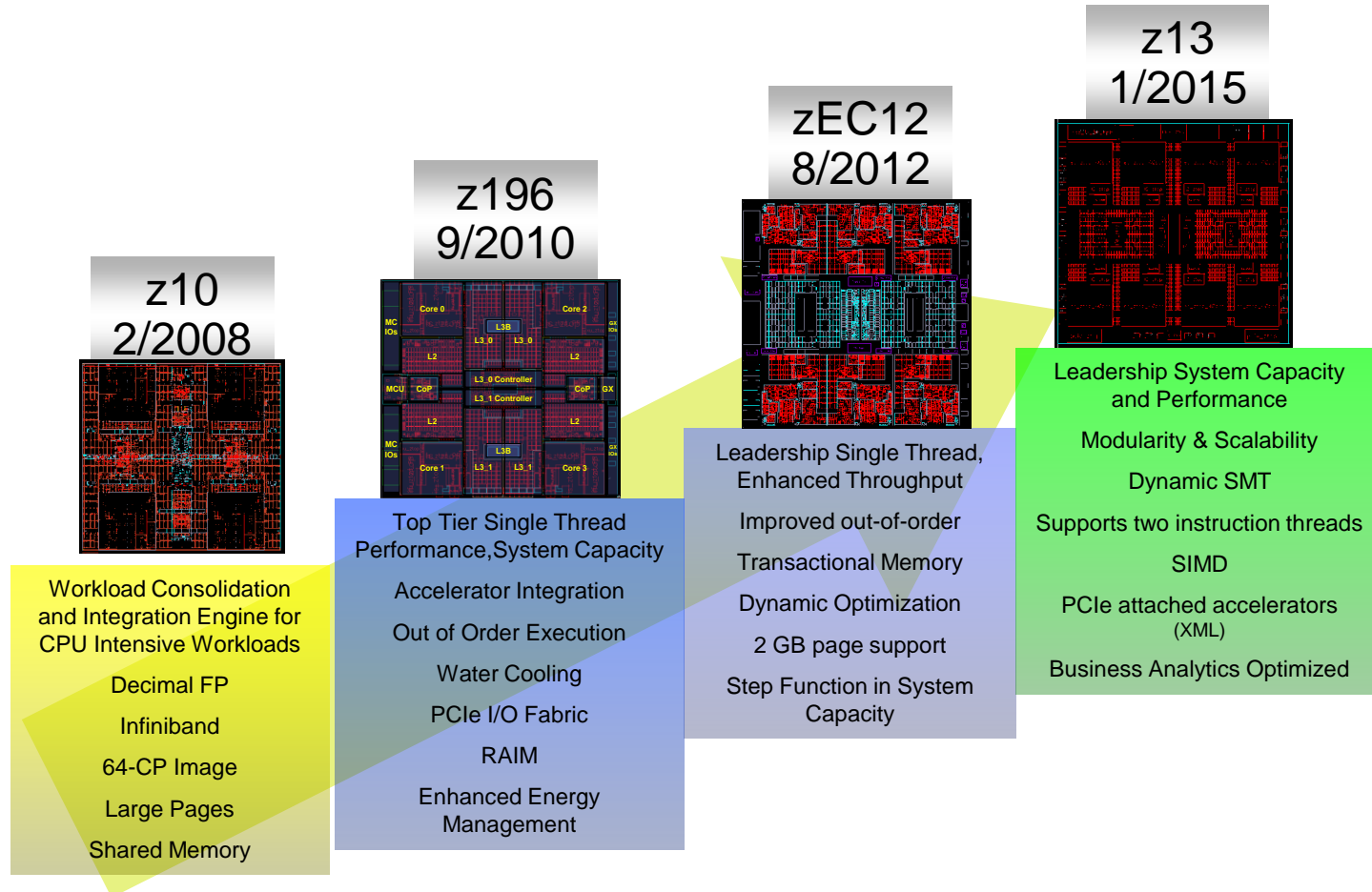


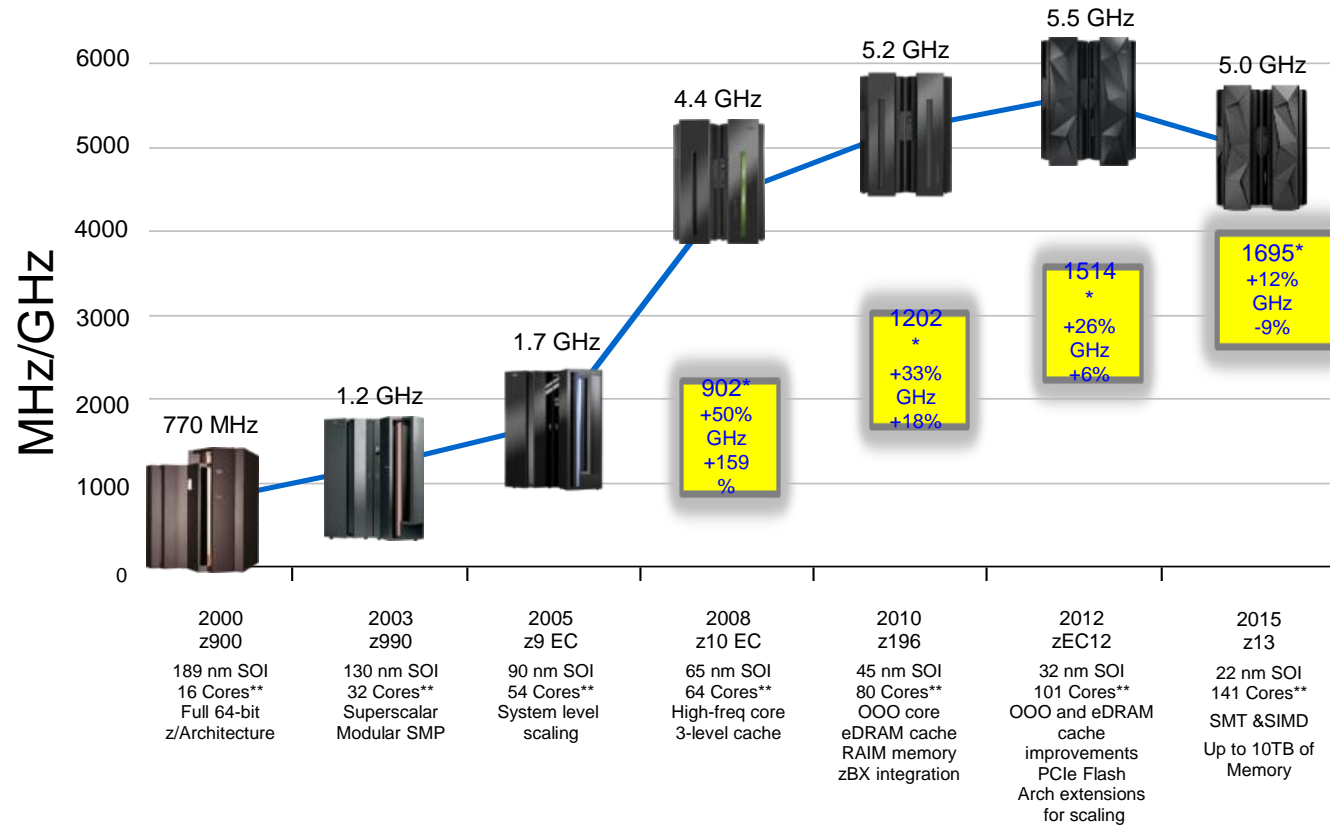
The IBM z13 SIMD Accelerators for Integer, String, and Floating-Point



z Systems - Processor Roadmap



z13 Continues the CMOS Mainframe Heritage Begun in 1994



* MIPS Tables are NOT adequate for making comparisons of z Systems processors. Additional capacity planning required

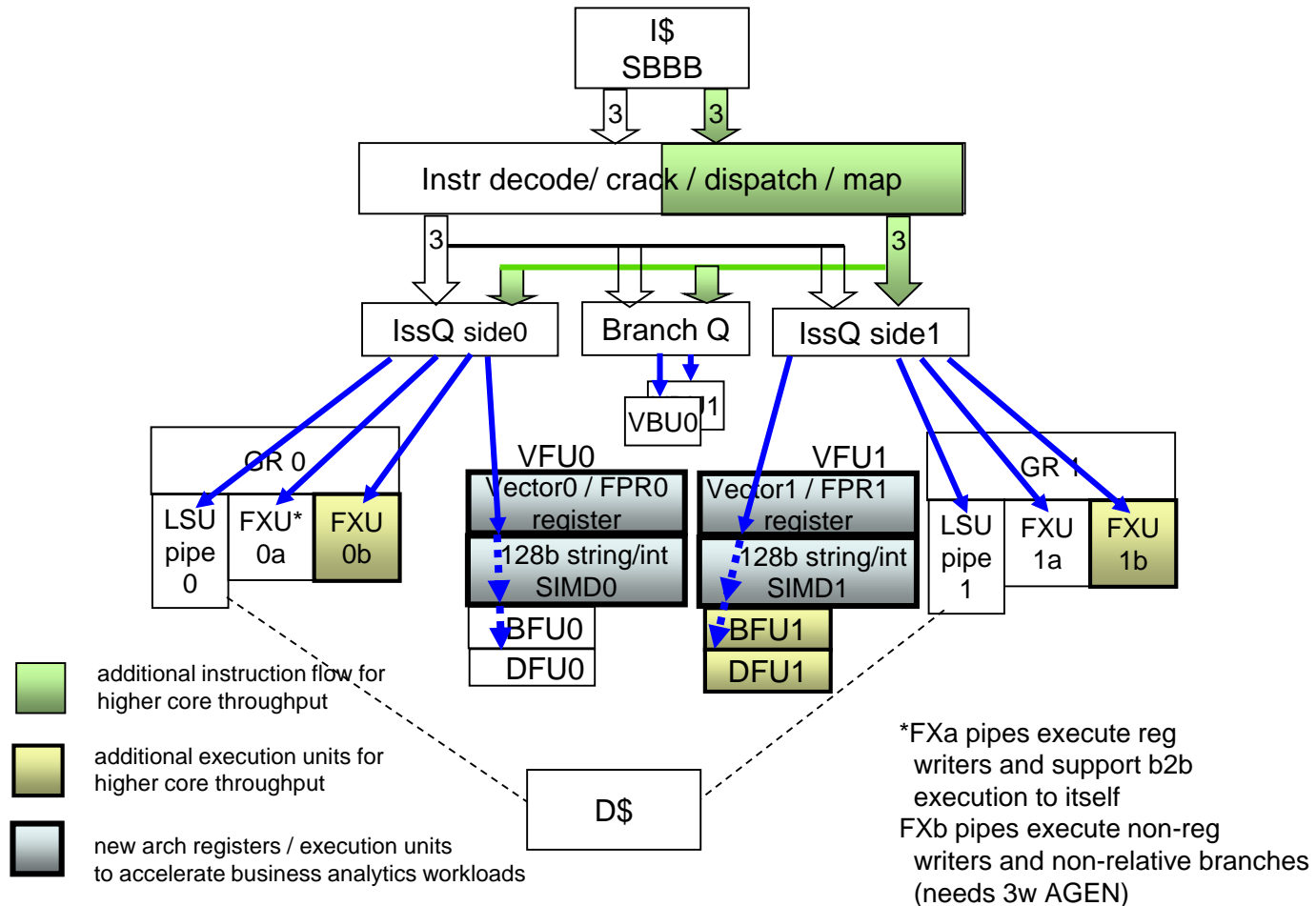
** Number of PU cores for customer use

z13 Core changes

- 1Q 2015 GA
- 8 cores per die
- 24 CP chips in system
- 192 cores x 2 threads => 384 threads
- Grow LSPR 8way by 11%, up to 1.4x perf with SMT2

- ST -> 2 way SMT
- Double the instruction fetch, decode, dispatch bandwidth
- Frequency hit (5.2 GHz z196 -> 5.6 GHz zEC12 -> 5.0 GHz z13)
- Double the number of execution units
 - 2 FXU writers, 2 FXU cc,
 - 2 BFU, 2 DFU
 - Quad precision is faster, Divide is faster
 - 2 SIMD units
 - 2 DFX – accelerate SS decimal

z13 Instr / Execution Dataflow



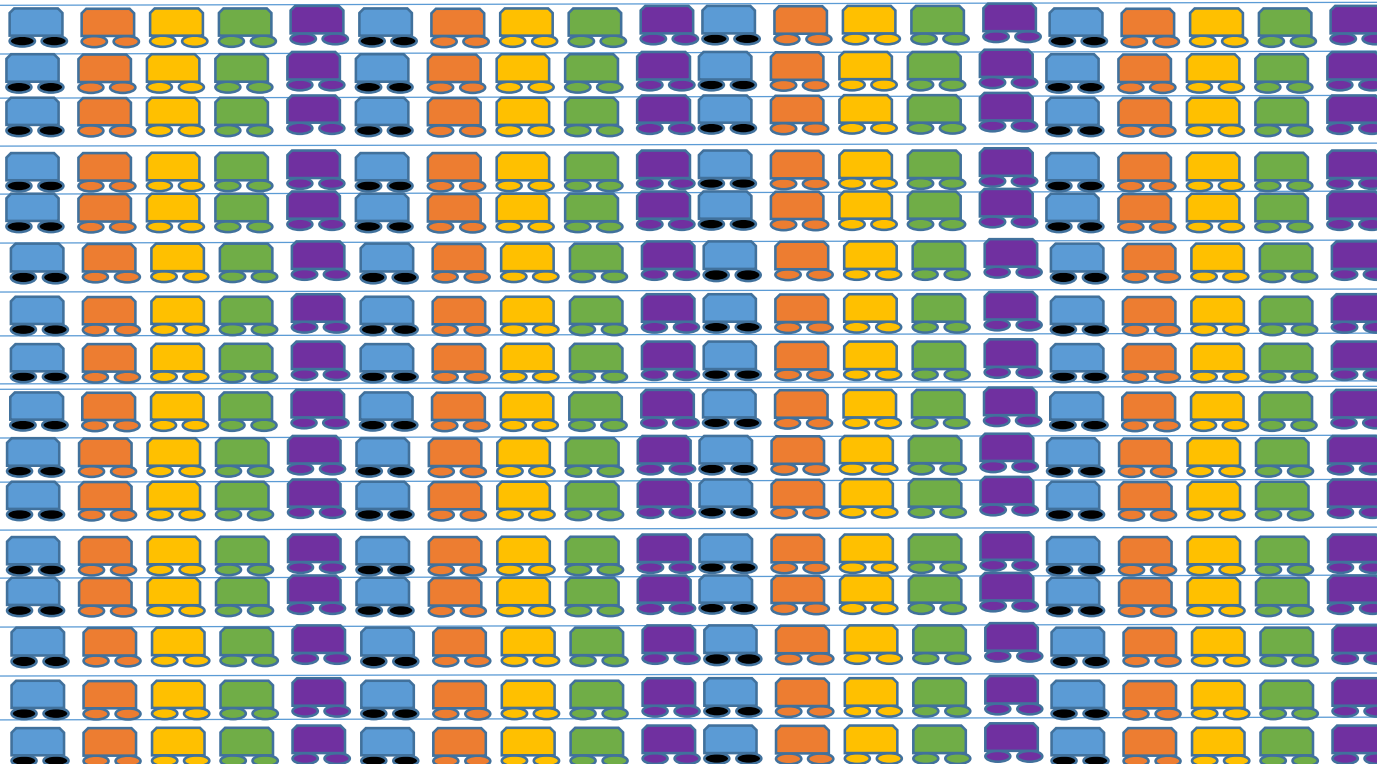
Performance

- Technology is not getting any faster
- System z is already the fastest at 5 GHz for the past couple generations
- Parallelism is the future
 - Simultaneous Multi-Threading – SMT
 - Single Instruction Multiple Data – SIMD
- Huge performance gains possible through parallelism
- Got to be smarter and figure out best ways to utilize parallelism

SIMD – Single Instruction Multiple Data

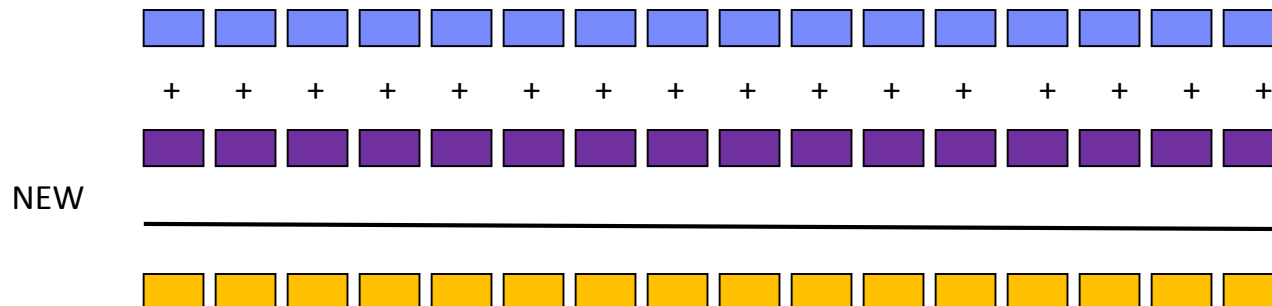


Old road



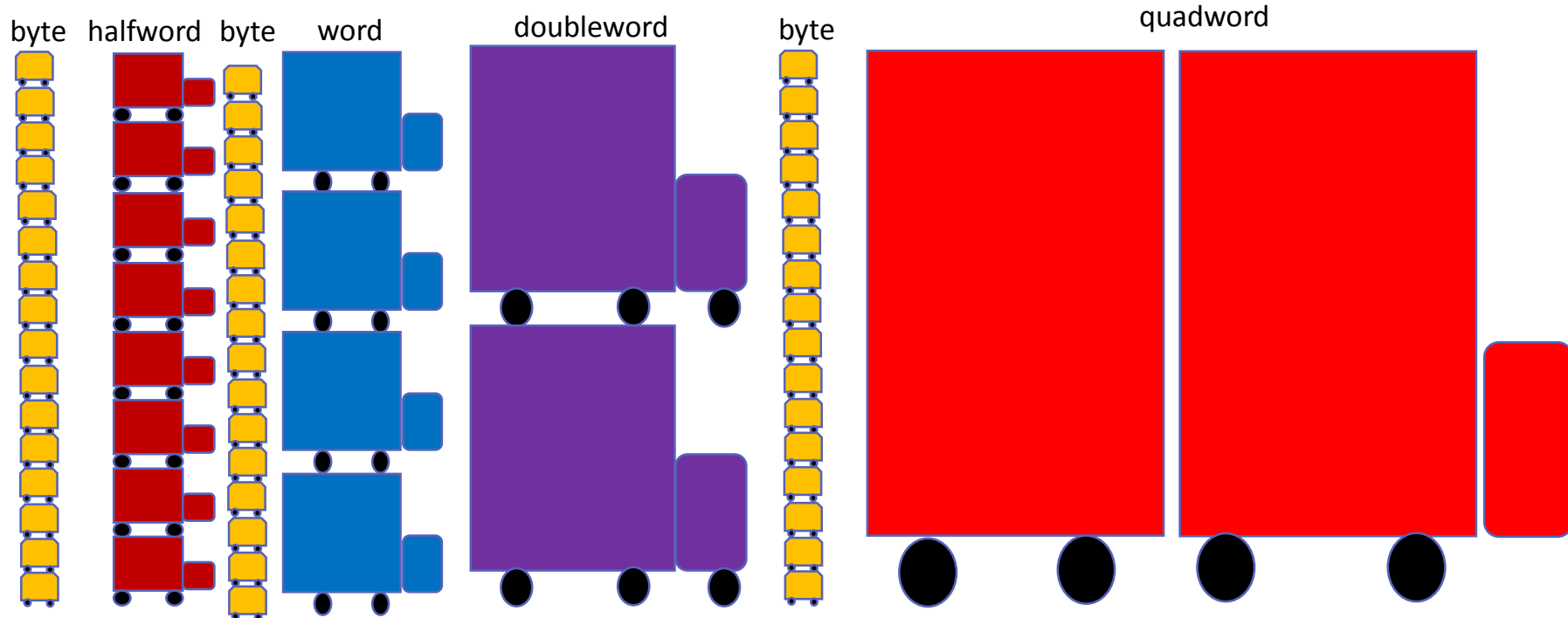
New super
highway

SIMD – Single Instruction Multiple Data



Different size data types – all 128 bit wide

- 16 x 8 , 8 x 16, 4 x 32, 2 x 64 or 1 x 128 vs 1x64b



Datatypes

1.F65438A890167 x 2⁽⁻³³¹⁾

-561

1,534,678,432. x 10⁽⁻³²⁾

"Hello World\0"

"To be, or not to be."

Data Types Supported

- Integers – unsigned and two's complement
- Floating-point – System z supports more types than any other platform
 - 32 (Single Precision - SP),
 - 64 (Double Precision – DP), and
 - 128 bit (Quad Precision – QP)
 - Radices of 2 (Binary Floating-Point – BFP),
 - 10 (Decimal – DFP),
 - 16 (Hexadecimal – HFP)
- Character Strings
 - Characters 8, 16, and 32 bit
 - Null terminated(C) and Length based (Java)

Z13 SIMD accelerates:

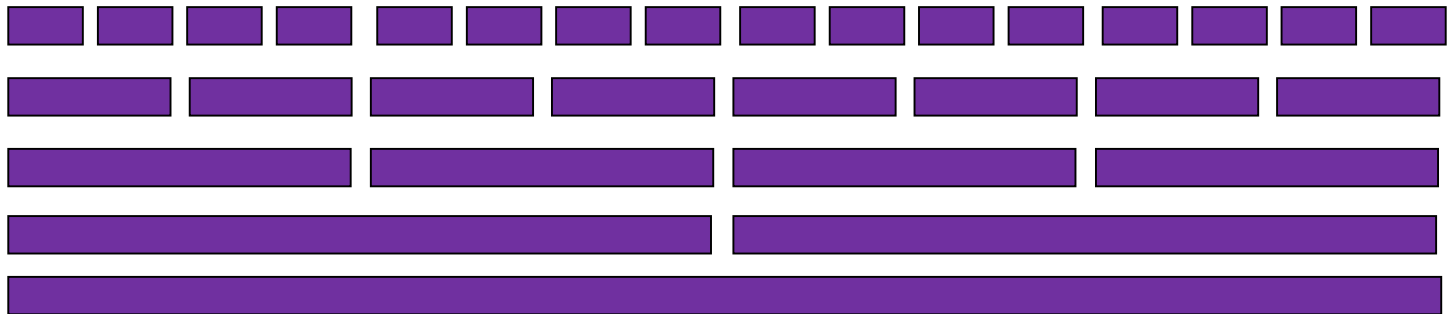
- Integer
- Binary Floating-Point Double Precision – BFP DP
 - Also indirectly quad precision
- Strings

SIMD Integer

- z13 has 2 new SIMD execution pipelines each 128 bits wide
- 16 x 8 bit / 8 x 16 bit / 4 x 32 bit / 2 x 64 bit / 1 x 128 bit
- Add, Subtract, Compare
- Logical operations
- Shifts
- Min / Max / Absolute Value
- Select
- CRC / Checksum

SIMD Integer - types

- Byte, Halfword, Word, Doubleword, Quadword in 128 bit register

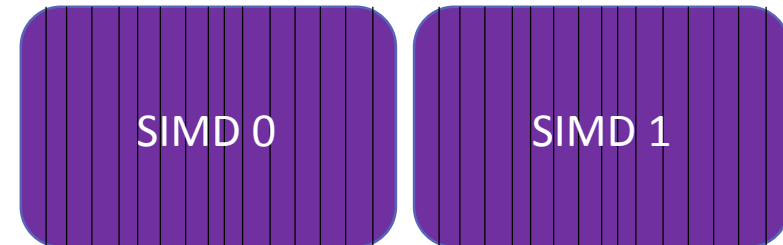
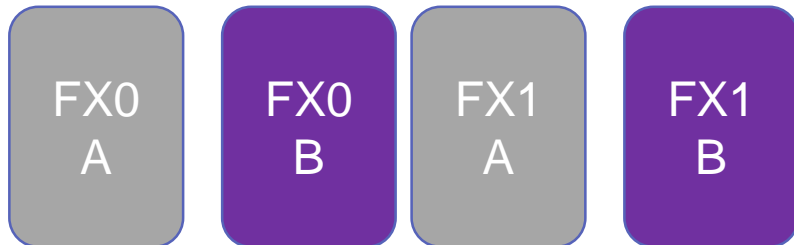
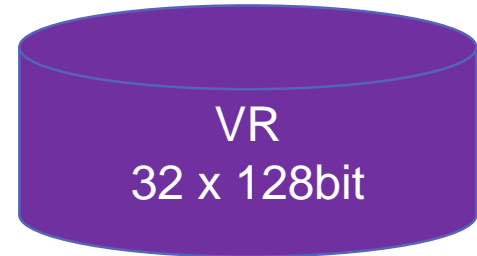


128b wide vector:

16xB, 8xHW, 4xW,

2xDW, 1xQW

zEC12 (grey) integer vs. z13 (grey + purple)



Increased from 2 x 64 bit pipelines
To 8 x 64 bit pipelines but even more
Parallelism with smaller integer datatypes

Integer Dataflows from zEC12 to z13

Integer Dataflow Roads are 4 times as wide supporting up to 18 times the number of elements



OLD 64 bit one lane road

OLD 64 bit one lane road

New 64 bit one lane road

New 64 bit one lane road

New 128 bit SIMD highway

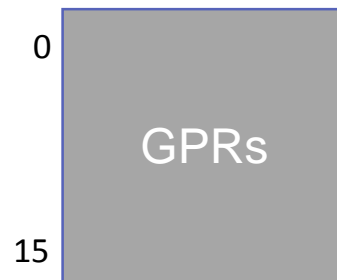
New 128 bit SIMD highway

Integer – Error Coding Acceleration

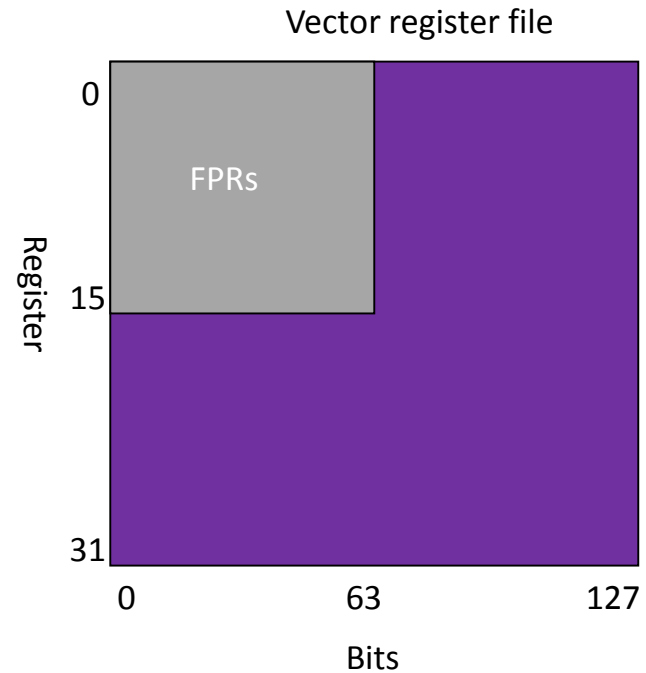
- Includes 64 bit Cyclic Redundancy Coding (CRC) assists to perform
 - $AxB + CxD + E$ of 64 bit in carryless form.
- Also includes a Checksum accelerator to add 5 x 32 bit operands modulo 2^{32} to accumulate a new 128 bit operand to a running sum every cycle.

More Execution Pipelines Need More Data

- Quadrupled the FPRs to create the Vector Register file
- VRs contains floats, integers, and strings



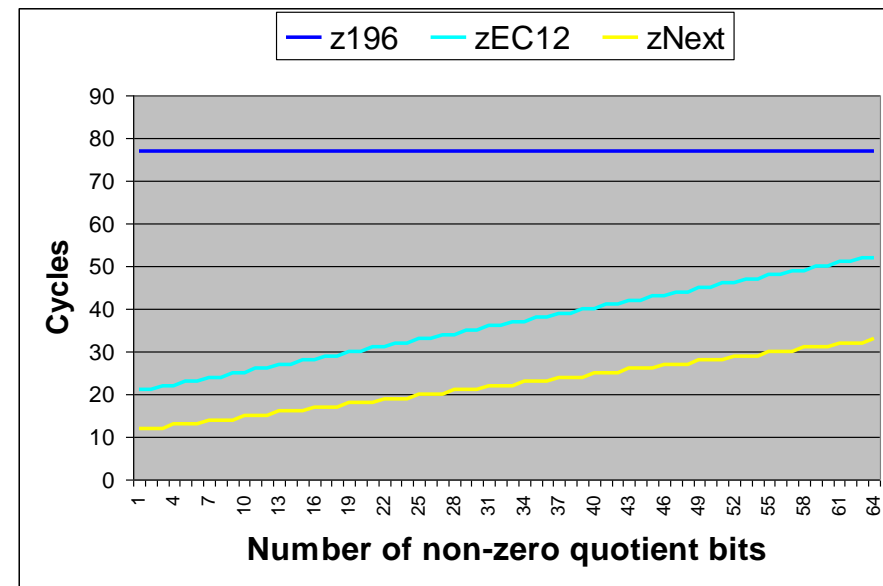
Think of this as a parking lot for rental cars
Or a truck depot.



Speeding Up Hash Functions / Integer Divide

- Z196
 - Multiplicative integer divide algorithm
 - Inside BFU, 1x, blocking
- zEC12
 - SRT based, 1st generation (k=2)
 - Speedup over z196: 1.4 – 3.6x
- z13
 - SRT based, 2nd generation (k=3)
 - Stand-alone engines in FXU, 2x
 - Non-blocking
 - Speedup over zEC12: 1.7x
 - Throughput: 3.4x

Latency of 64b Integer Divide

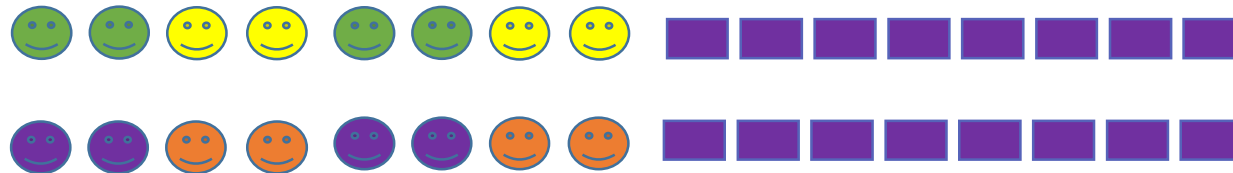


$$\begin{aligned} \text{Ex.: } 234876 \div 2 &= 78292 \\ 234876 \div 7893 &= 29 \end{aligned}$$

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 we double pump a vector to the same BFU pipeline

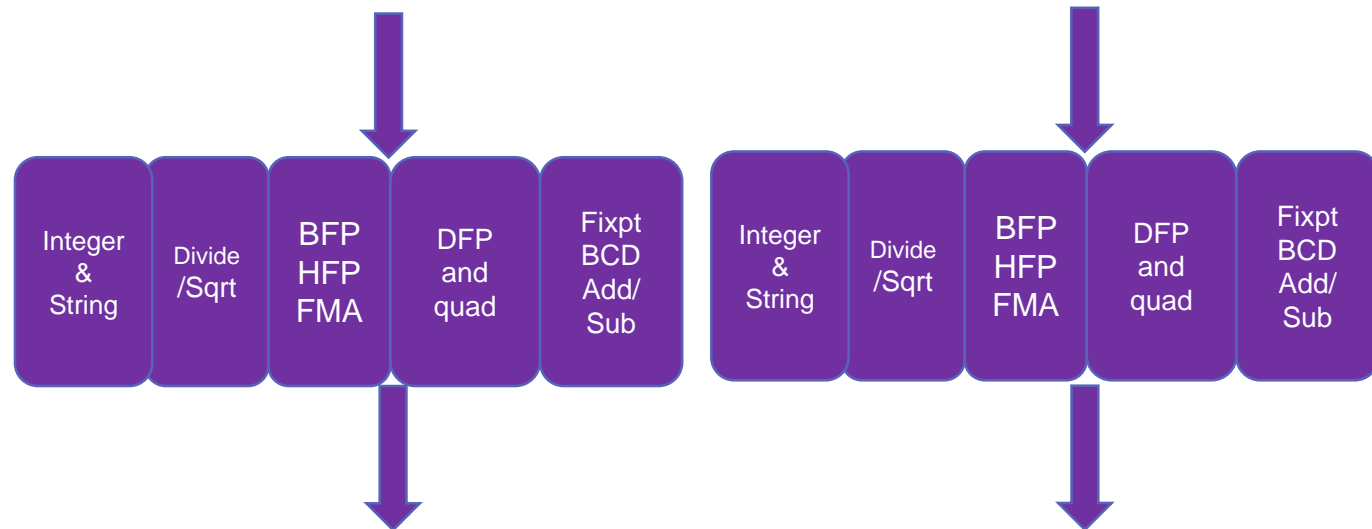
Improved Binary Floating-Point (DP, SP) Latencies

	z196 / zEC12	z13
Add, sub, mul, FMA, convert	8	8
Compare	8	3
Divide 32b	33	17
64b	40	27
SQRT 32b	39	21
64b	53	36

Z13 allows multiple floating point units to execute at same time and does not stall on long operations like divide and square root.



zEC12

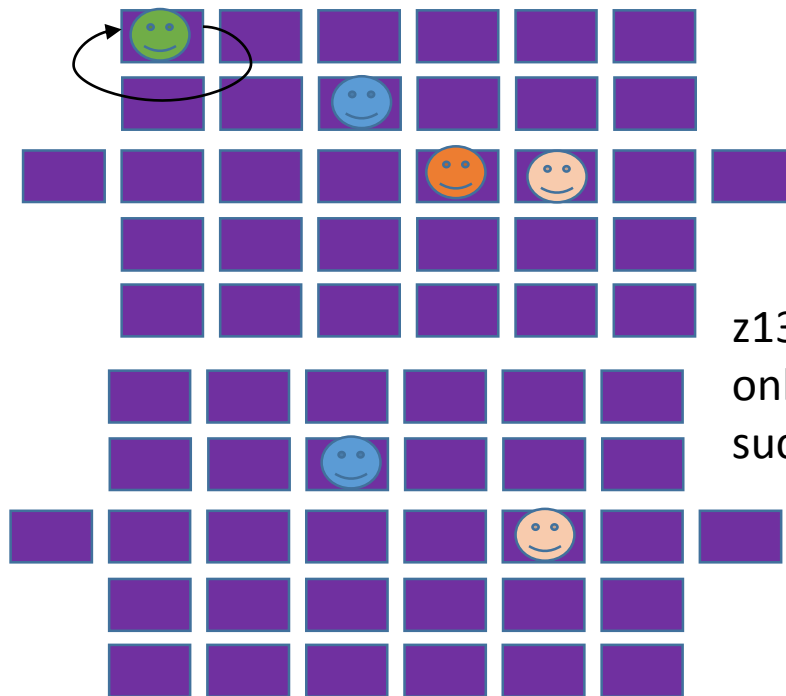


z13

z13 does not stall on divide, each floating-point pipeline now consists of 5 sub-pipelines.



zEC12 has 1 pipeline
Run-on instructions
Block pipeline

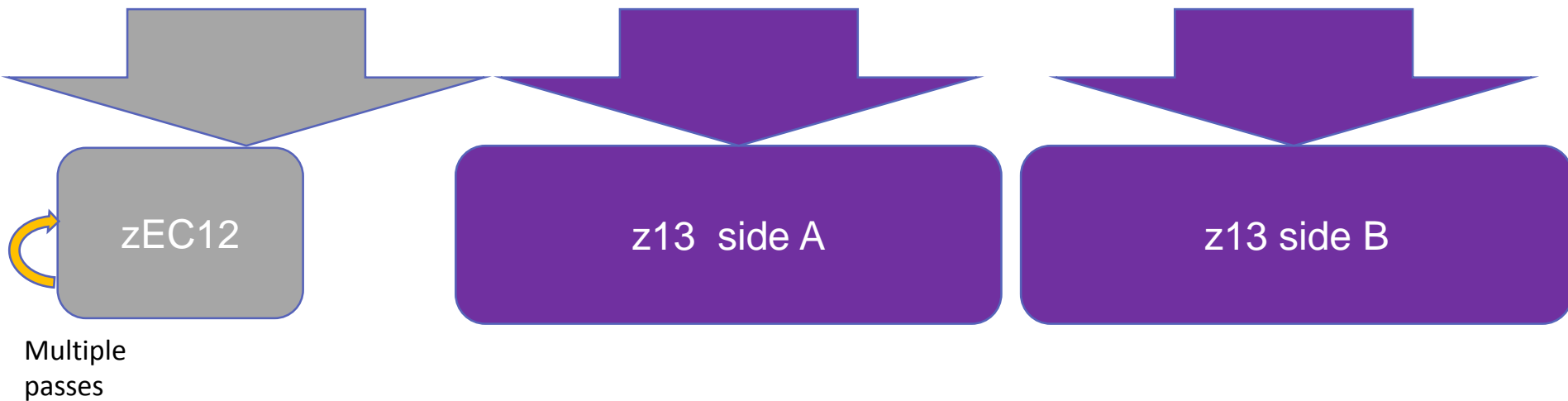


z13 has non-blocking pipelines,
only blocks on specific resource /sub pipeline,
such as divider



Floating-point Quad precision is much faster

- All other platforms use software, z is only platform to use hardware
- Moved quad precision to wide pipeline
- Now over 3x faster with 2x the bandwidth for a total of 6x performance over zEC12
- Also non-blocking



BFP Quad Precision latency in cycles

	zEC12	z13
add, sub	35	11
multiply	55 - 97	23
divide	~165	49
sqrt	~170	66

1 engine

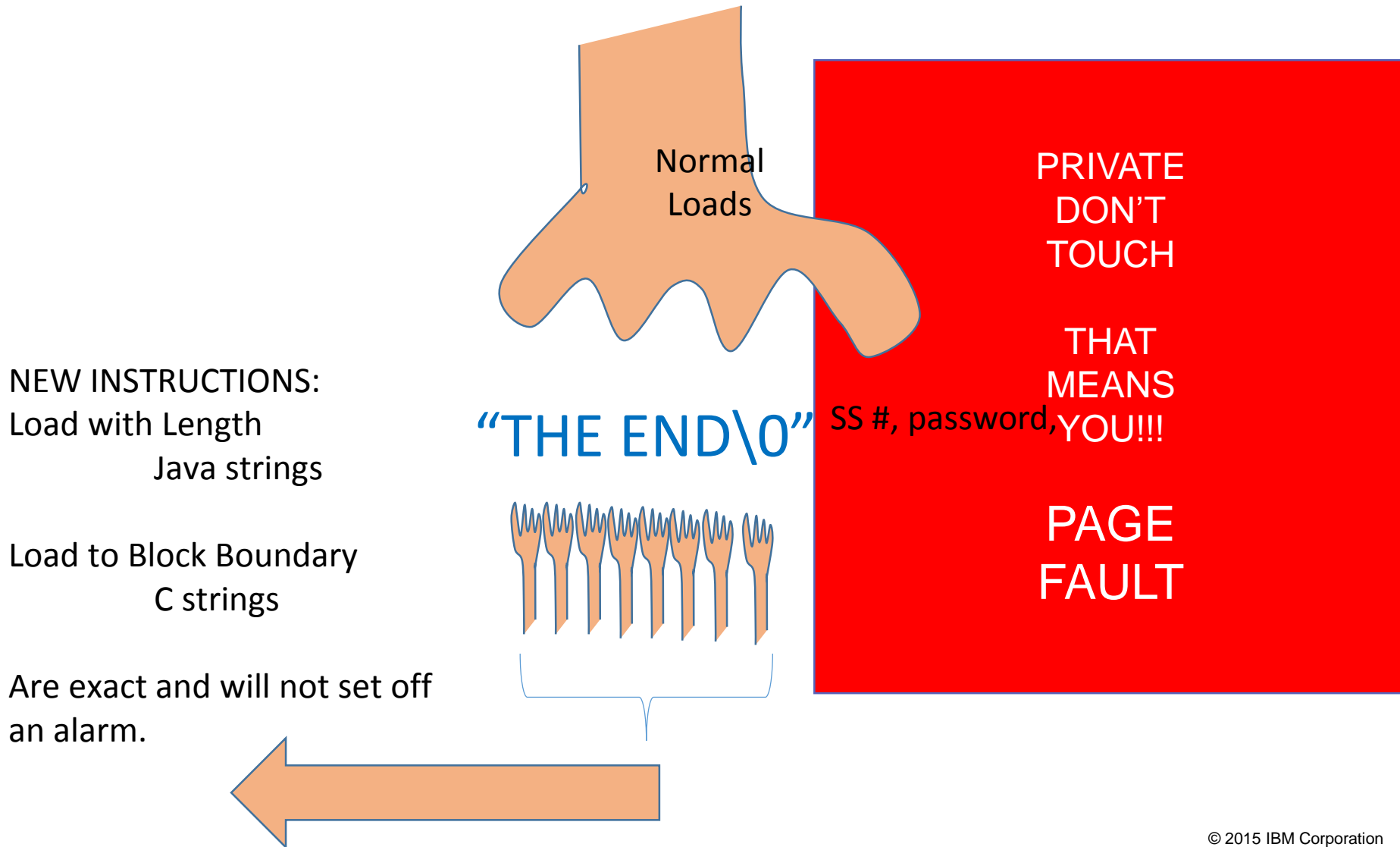
2 engines

Each 3 X faster

String Acceleration

- Character types supported are Byte, Half Word, and Word
- Loads and Stores for both
 - known length strings (Java strings), and
 - those with a null terminating character (C strings)
- String Search Acceleration
 - Find Equal
 - Find Not Equal
 - Range Compare

In the past, loading or storing to a string could result in exceptions

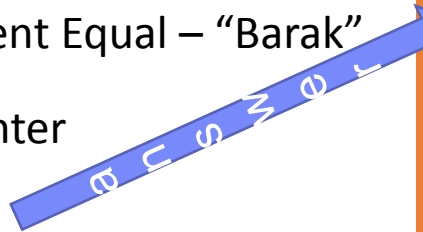


String Acceleration with Powerful New Instructions

16 characters of the string can be
Examined every cycle
Search text can be 16 characters
Or 8 pairs of characters for ranges

Vector Find Element Equal – “Barak”

Returns a Pointer



John Doe	123-45-6789	845-555-1212
Barak Hussein Obama II	987-65-4321	212-555-1212
Mary H Lamb	555-55-5555	555-555-5555
Humpty Dumpty	666-66-6666	666-666-6666
Kings Men	777-77-7777	777-777-7777
Emergency	911-91-1911	???-???-?????

Vector String Range Compare

Use it to Find IsNotAlphaNumericORHyphen

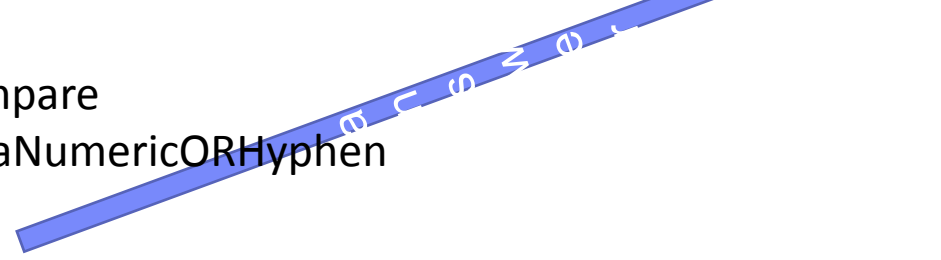
(GE '0' LE '9') or

(GE 'a' LE 'z') or

(GE 'A' LE 'Z') or

(EQ '-' EQ '-') ;

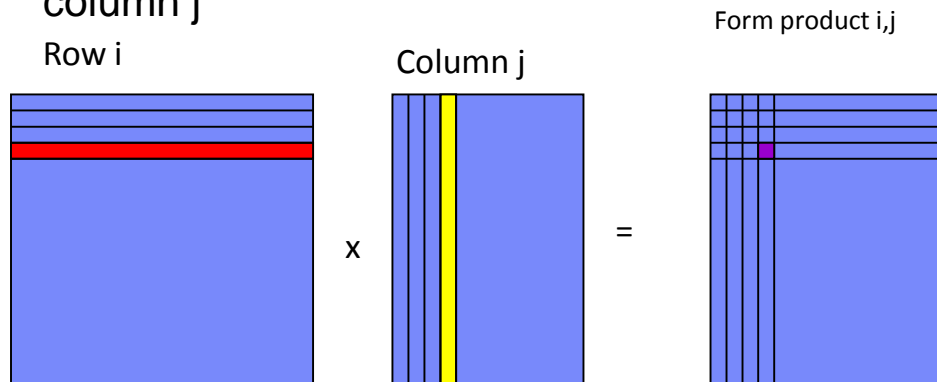
Returns a pointer



MATRIX MULTIPLICATION

Matrix multiply - DGEMM

- A product matrix element $p(i,j)$ is equal to the sum of a row i x column j



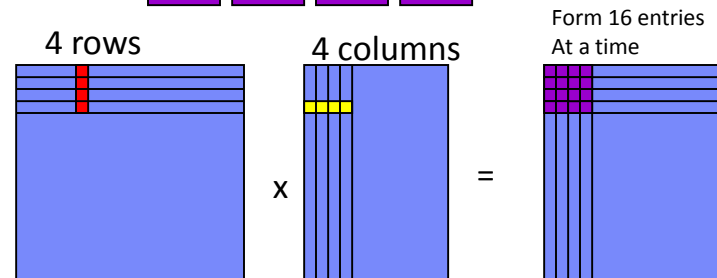
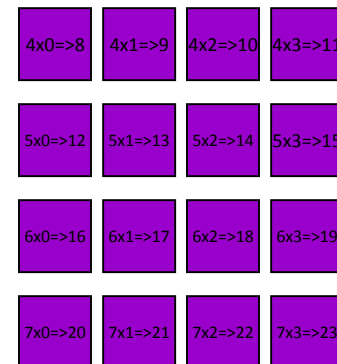
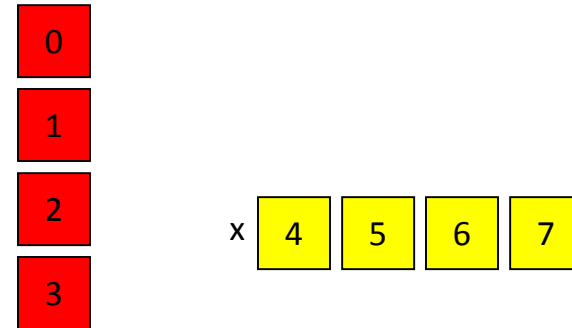
To create independent operations we do the opposite,
We multiply a column by a row and form independent product elements

FPRs are allocated to accumulated product elements (in green)

An iteration consists of accumulating one new product to a set of independent Product elements.

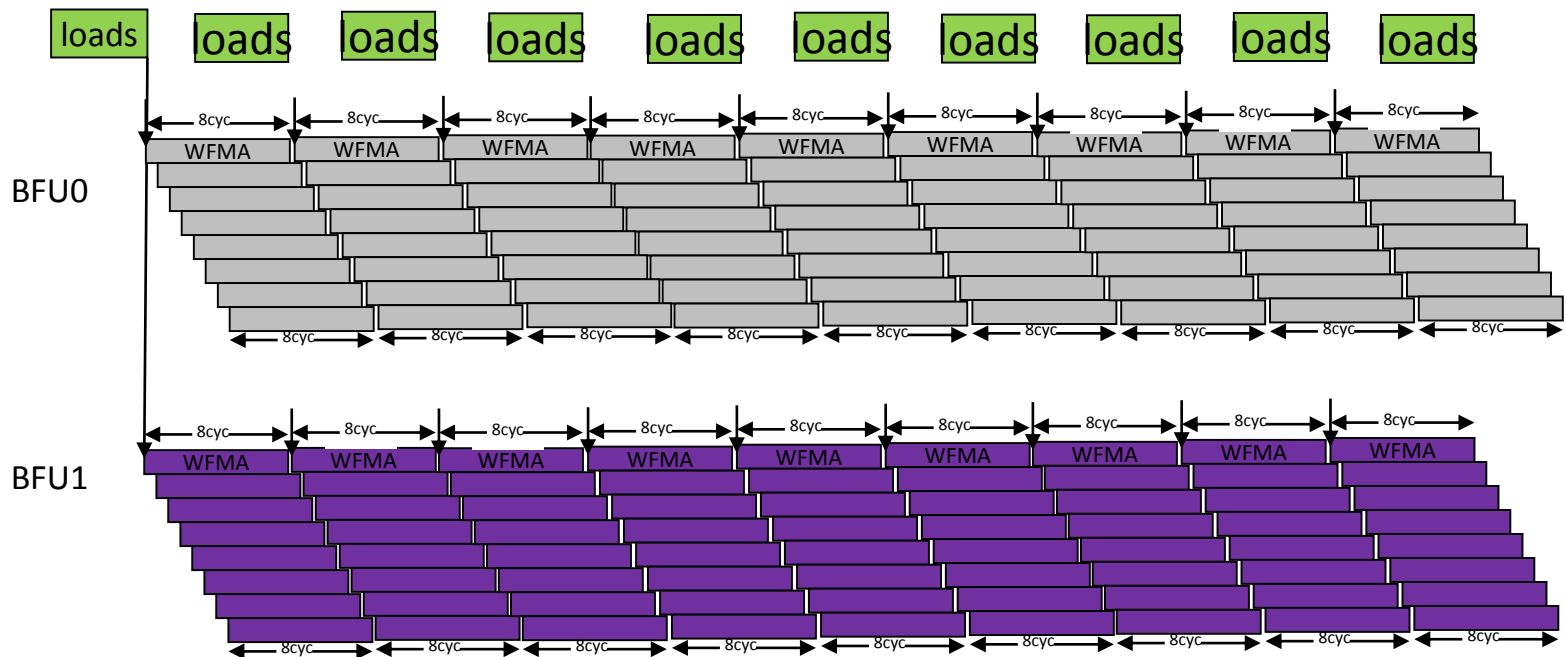
DGEMM 4x4

- Load 8 FPRs
- 16 FMAs
 - With 2 BFU pipes
 - Could separate by 8 cycles
 - Need 24 arch FPRs (use Vector Scalar)
- # Regs = $N \times M + N + M$, could do 4x5 with 32 regs but not power of 2
- Other algorithms could separate the adds to not delay multiplies



DGEMM 4x4 Vector-Scalar

No stall cycles



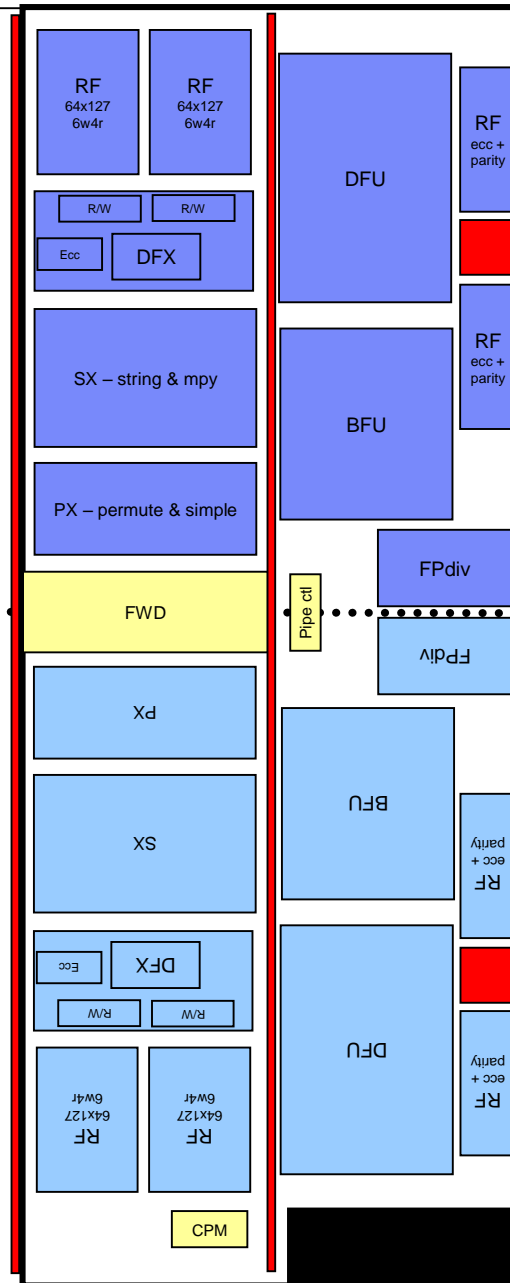
32

16 independent FMAs

TIME

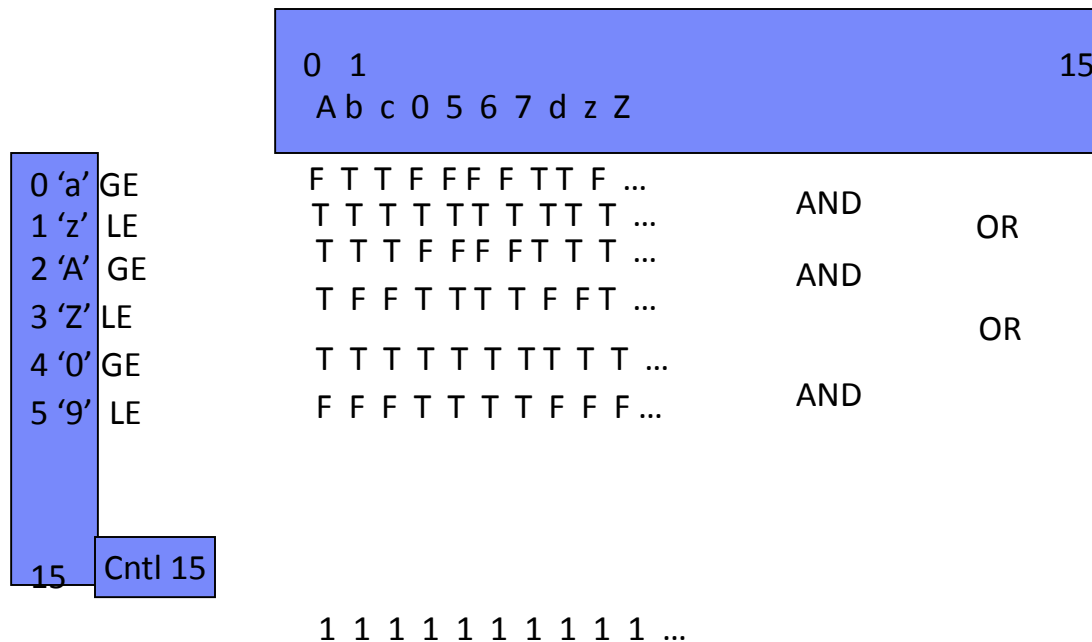
DGEMM Conclusions

- So 2x4 algorithm about optimal for 16 regs and only 1 BFU needed.
- To use bandwidth of 2 BFUs with 8 cycle latency then 4x4 needed. This would approximately double the throughput.
 - Can form 16 entries in same time to form 8.
- Use of vectorized 4x8 causes no slow down to code now, and in the future may allow for up to 2x performance.



Backup / Old slides

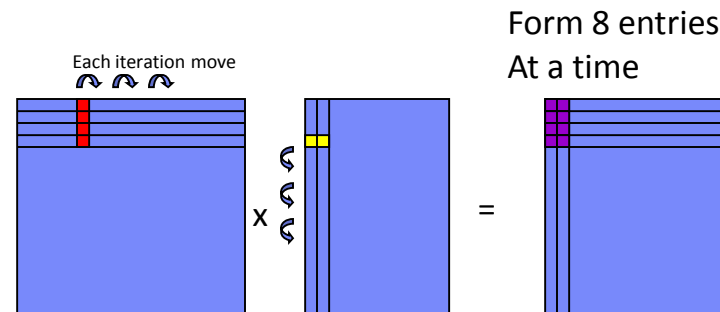
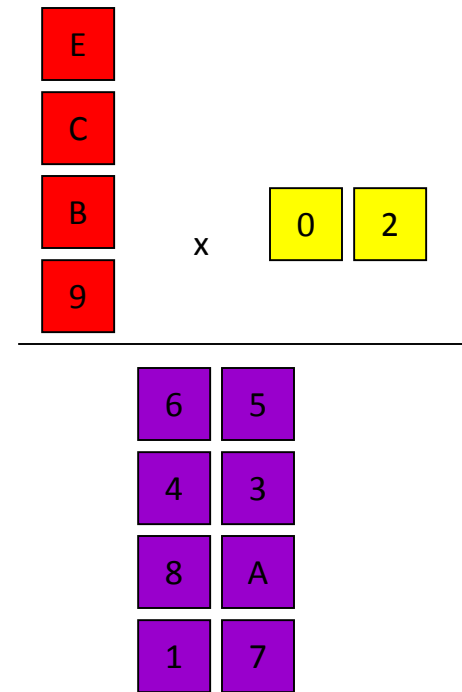
Range compare for isAlphaNumeric() in one instruction



DGEMM 2x4

This is the actual sequence used:

- Load 0,2,9,E,C,B
- $A = A + 2xB$
- $8 = 8 + Bx0$
- $7 = 7 + 2x9$
- $5 = 5 + 2xE$
- $3 = 3 + 2xC$
- $1 = 1 + 0x9$
- $6 = 6 + Ex0$
- $4 = 4 + Cx0$



2x4 scroll pipes

- First iteration
 - 6 Loads come back in different cycles
 - So FMAs start up randomly
- Further iterations
 - Loads done ahead of time
 - Critical path is FMA to FMA for accumulation

FMA's Dependent due to Accumulation

17437	10062DDDDD..mIIIIIIILLLLLLpIII.....NNNN.r.....	17437	q	80001c0a	LD	NV	68E5B000	Loads are done early
17438	10063DDDDD.mIIIIIIl11111pIII.....NNNN.r.....	17438	r	80001c0e	LD	NV	68C4B000	
17439	10063DDDDD.mIIIIIIl11111pIII.....NNNN.r.....	17439	s	80001c12	LD	NV	68B3B000	
17440	10066DDDDD.mIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFpINNNN.r.....	17440	t	80001c16	MADBR	NV	B31EA02B	
17441	10066DD..mIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFpIIII.NNNN.r.....	17441	u	80001c1a	MADBR	NV	B31E80B0	
17442	10067DD..mIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFpINNNN.r.....	17442	v	80001c1e	MADBR	NV	B31E7029	
17443	10068DD..mIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFpINNNN.r.....	17443	w	80001c22	MADBR	NV	B31E502E	
17444	10069D....mIIIIIIIIIIIIIIIIIIIIfffffffpINNNN.r.....	17444	x	80001c26	MADBR	NV	B31E302C	
17445	10069D....mIIIIIIIIIIIIIIIIIIIIfffffffpIIII.NNNN.r.....	17445	y	80001c2a	MADBR	NV	B31E1009	
17446	10069D....mIIIIIIl11111pIII.....NNNN.r.....	17446	z	80001c2e	LD	NV	682BA008	
17447	10069DD....mIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFpIIII.NNNN.r.....	17447	A	80001c32	MADBR	NV	B31E60E0	
17448	10069DD....mIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFpIIII.NNNN.r.....	17448	B	80001c36	MADBR	NV	B31E40C0	
17449	10069DD....mIIIIIIILLLLLLpIII.....NNNN.r.....	17449	C	80001c3a	LD	NV	6892B008	
17450	10070DDDDD..mIIIIIIILLLLLLpIII.....NNNN.r.....	17450	D	80001c3e	LD	NV	6801B008	
17451	10070	d.....DDDDD..mIIIIIIILLLLLLpIII.....NNNN.r.....	17451	E	80001c42	LD	NV	68EB5008	
17452	10070	d.....DDDDD..mIIIIIIILLLLLLpIII.....NNNN.r.....	17452	F	80001c46	LD	NV	68CB4008	
17453	10070	d.....DDDDD.mIIIIIIl11111pIII.....NNNN.r.....	17453	G	80001c4a	LD	NV	68BB3008	
17454	10074	d.....DDDDD.mIIIIIIIIIIIIIIIIIIIIIIfffffffpINNNN.r.....	17454	H	80001c4e	MADBR	NV	B31EA02B	FMA dep on Prior FMA
17455	10074	d.....DDDDD.mIIIIIIIIIIIIIIIIIIIIIIfffffffpIIII.NNNN.r.....	17455	I	80001c52	MADBR	NV	B31E80B0	
17456	10075	d.....DDD.mIIIIIIIIIIIIIIIIIIIIIIfffffffpINNNN.r.....	17456	J	80001c56	MADBR	NV	B31E7029	
17457	10076	.ddd.....DDD.mIIIIIIIIIIIIIIIIIIIIIIfffffffpINNNN.r.....	17457	K	80001c5a	MADBR	NV	B31E502E	
17458	10077	.ddd.....DDD.mIIIIIIIIIIIIIIIIIIIIIIFFFFFFFFFFFpINNNN.r.....	17458	L	80001c5e	MADBR	NV	B31E302C	
17459	10077	.ddd.....D....mIIIIIIIIIIIIIIIIIIIIIIfffffffpIIII.NNNN.r.....	17459	M	80001c62	MADBR	NV	B31E1009	
17460	10077	.ddd.....D....mIIIIIIIIIIIIIIIIIIIIIIfffffffpIIII.NNNN.r.....	17460	N	80001c66	MADBR	NV	B31E60E0	
17461	10077	.ddd.....D....mIIIIIIIIIIIIIIIIIIIIIIfffffffpIIII.NNNN.r.....	17461	O	80001c6a	MADBR	NV	B31E40C0	
17462	10078	.ddd.....D....mIIIIIIIIIIIIIIIIIIIIIIfffffffpIIII.NNNN.r.....	17462	P	80001c6e	AGHI	NV	A7BB0010	

SCROLL PIPELINES

D-decode, m-mapping, l-issue, F-fpu execution, p-putaway, r-checkpointed

DGEMM 2x4

