How Do You Do What You Do When You're a z10 CPU?

Bob Rogers
IBM Corporation
rrrogers@us.ibm.com

SHARE in Boston Summer 2010
Session #7534

SHARE in Boston

Trademarks

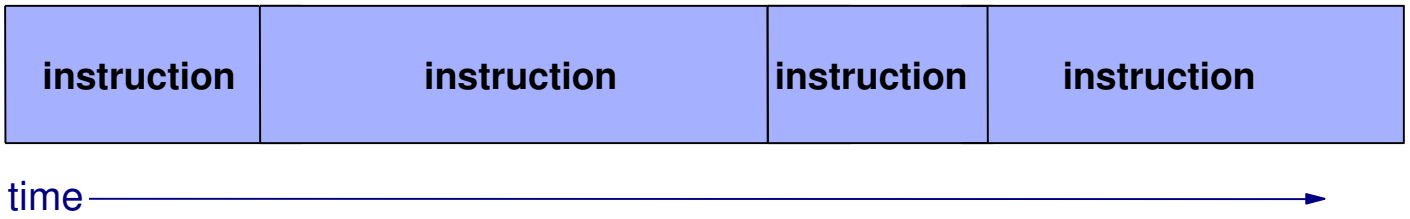
z/OS
zSeries
z/Architecture
IBM®

Topics

- Overview of instruction Processing
- What's different about z10
- Superscalar Grouping
- The Pipeline and its Hazard
- Branch Prediction
- Cache Topology
- Coprocessors
- TLB2 and Large Pages

Conceptual View of Execution

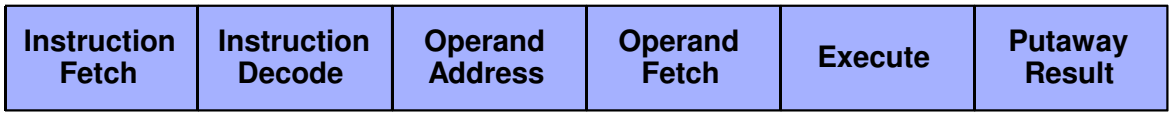
Instructions are executed in the order they are seen.
Every instruction completes before the following instruction begins.
Instructions take a varying amount of time.
Instructions have direct and immediate access to main storage.



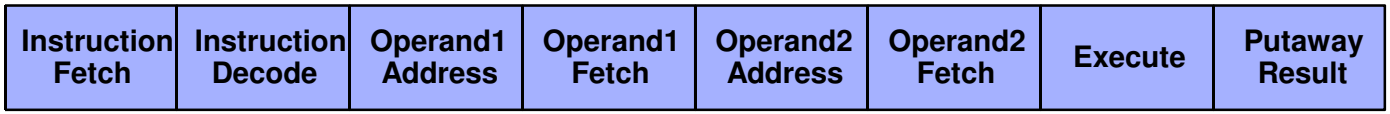
But, this is an illusion.

Pipeline View of Instructions

Individual instructions are really a sequence of dependent activities, varying by instruction:



for example: `A R1,D2(X2,B2)`

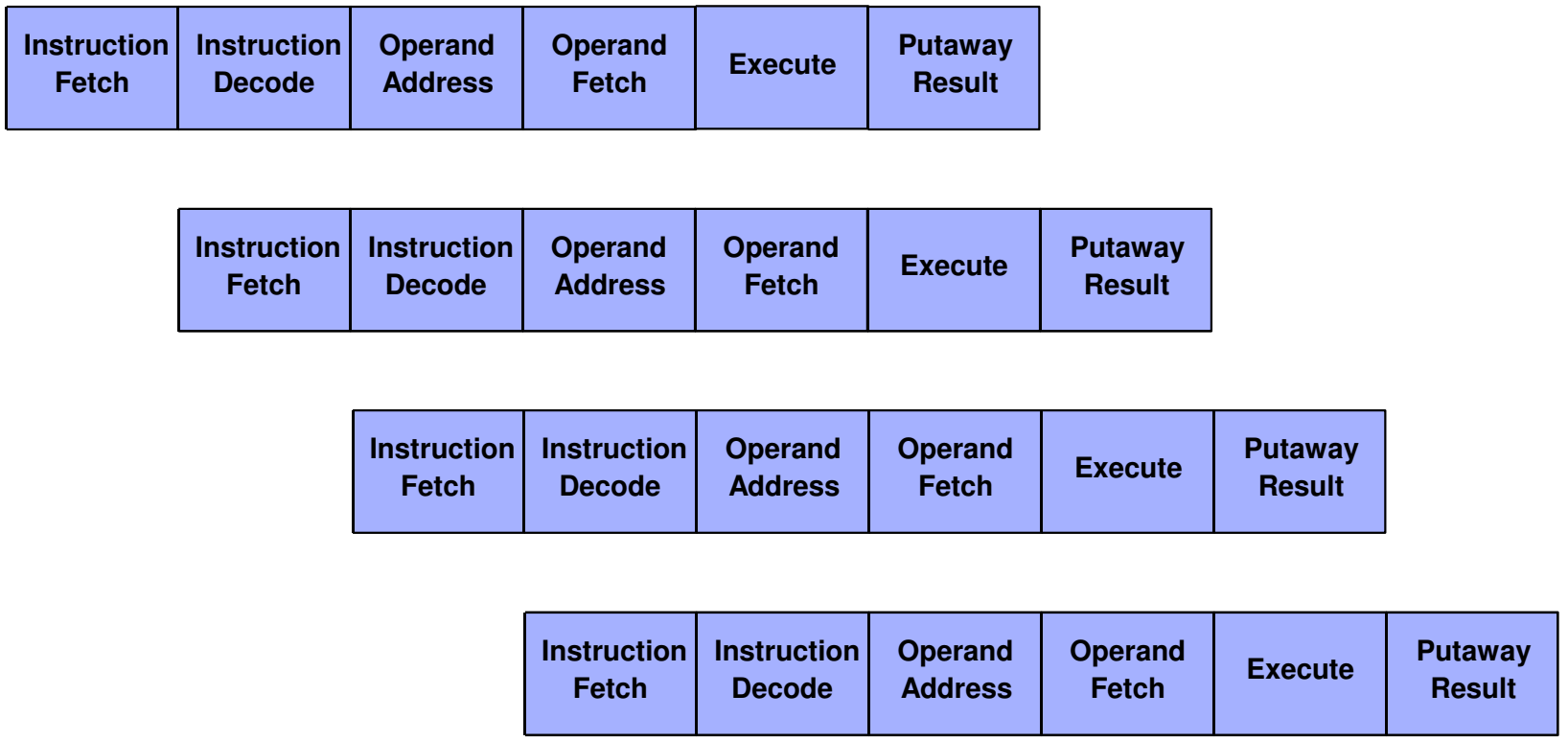


for example: `CLC D1(L,B1),D2(B2)`

for example: `UPT` **(Update Tree)**

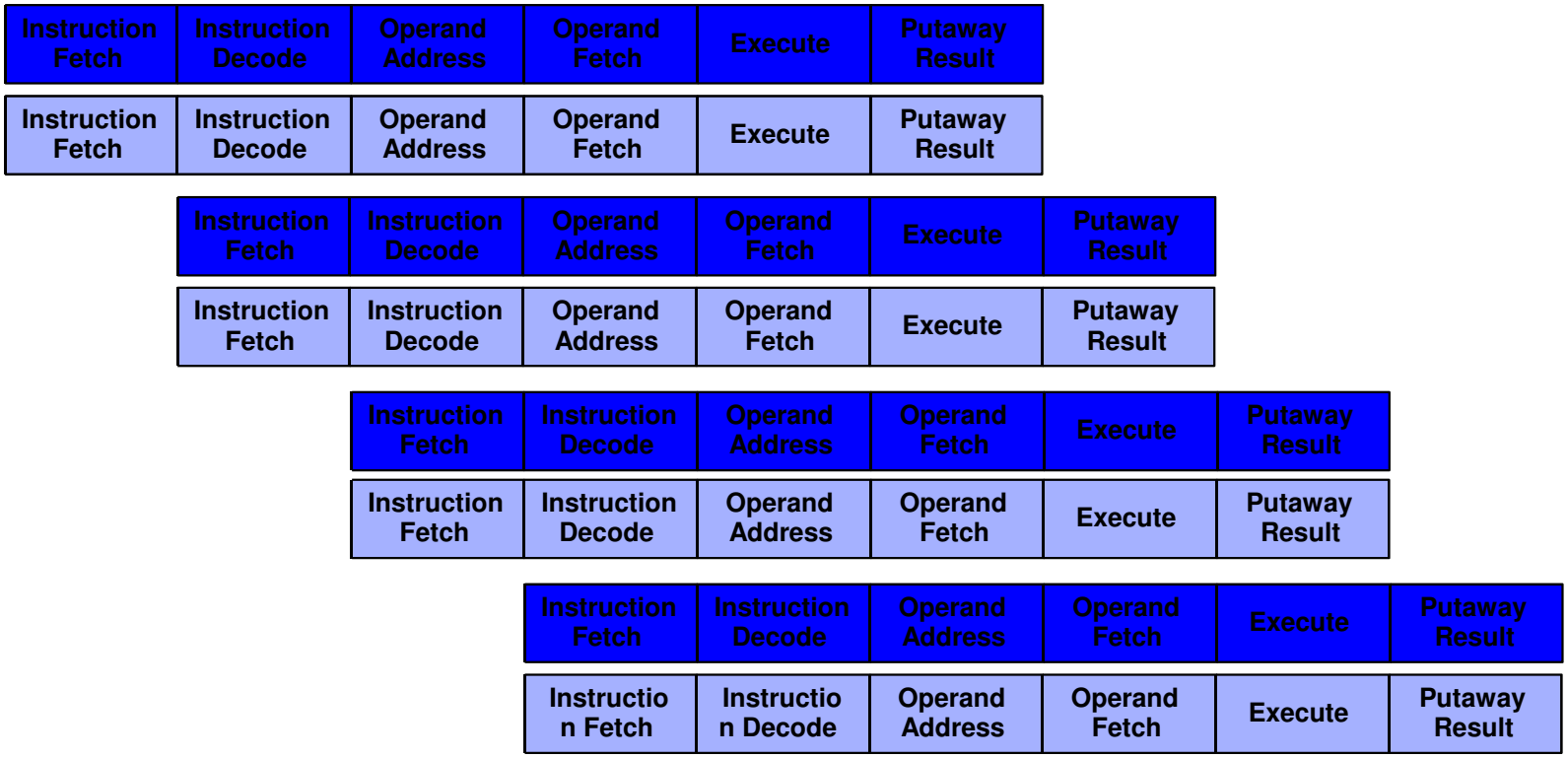
Pipeline View of Instructions

Each stage in the execution of an instruction is implemented by distinct components so that execution can be overlapped.



Superscalar multiple instruction overlap

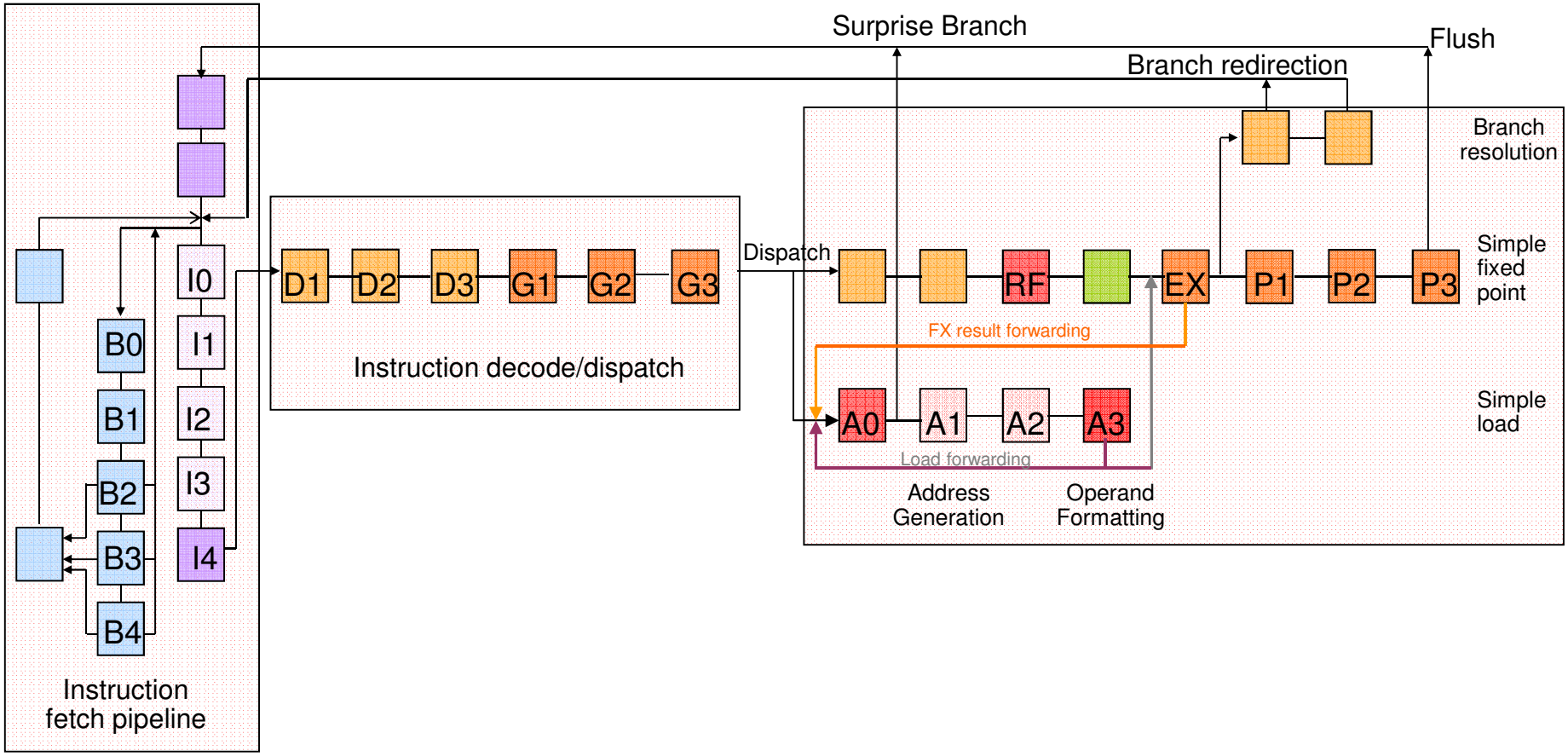
A Superscalar processor can process multiple instructions simultaneously because it has multiple units for each stage of the pipeline. But, the apparent order of execution is still maintained.



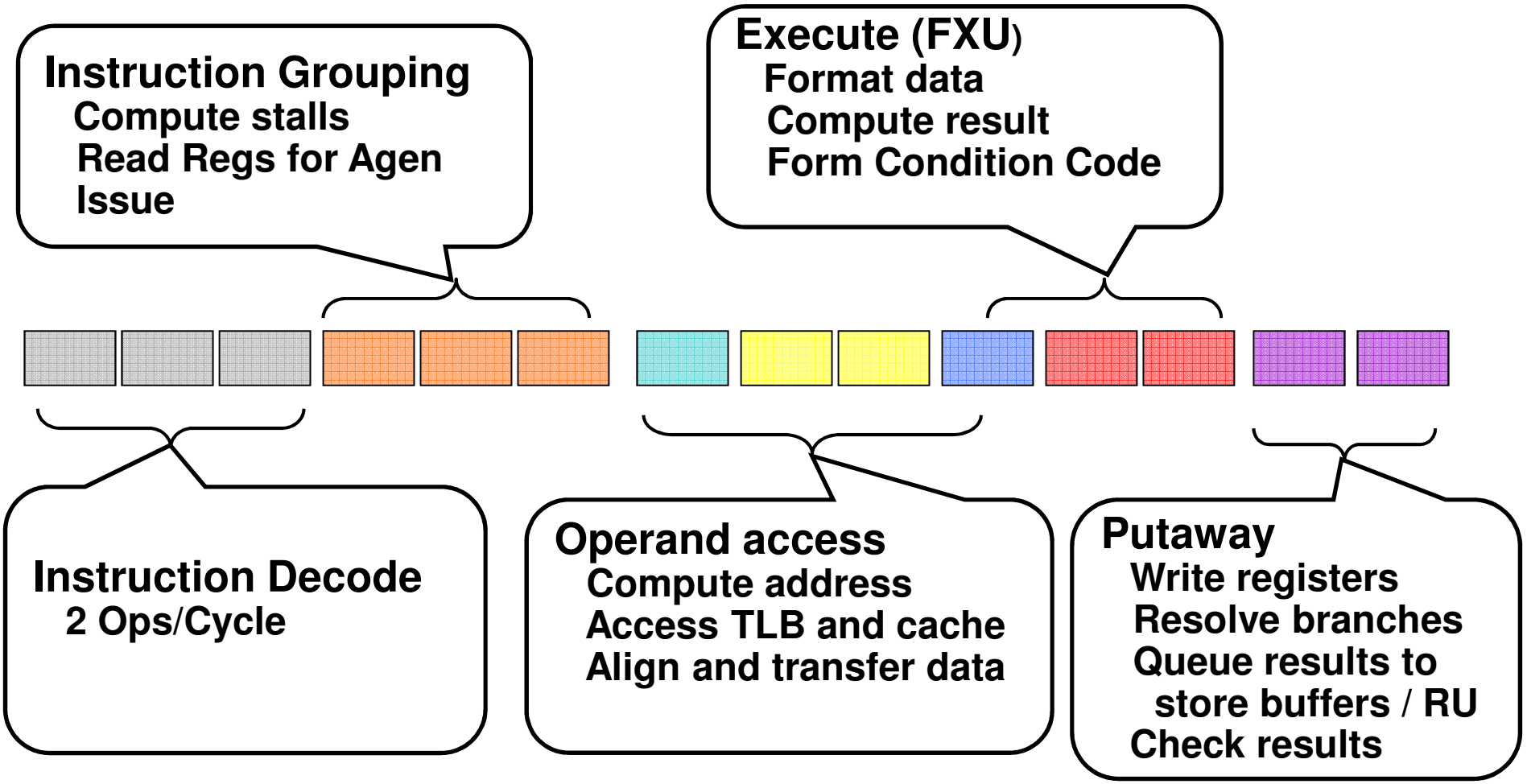
The IBM System z10 compared to z9

- Z10 has a radically different instruction processor
  - *high frequency* processor
    - 4.4 GHz vs 1.7GHz (2.5x)
  - much longer instruction pipeline
    - 14 stages vs 6 stages
  - different type of instruction pipeline
    - Rejecting pipeline vs stalling pipeline
    - Reject-recylce cost about 9 cycles
  - still performs in-order execution
  - still favors RX instructions

System z10 Instruction Pipeline (partial)



Core Pipeline



Superscalar Grouping Rules

- Most single-cycle instructions are “superscalar”
- Instruction groups contain 1 or 2 superscalar instructions
- First or Last instruction can be a branch instruction
- Instruction groups are held in decode dispatch unit to avoid pipeline hazards like AGI and OSC
- Some instructions that were superscalar on z9 are not superscalar in z10

High frequency is great, but….

- There are some negative affects cause by the short cycle time. For example:
  - Some instructions can no longer be done in the shorter cycle time and now take more than one cycle
    - Most instructions that involve sign propagation (e.g. LH) are no longer single cycle
  - Requiring both the true and complement value of a register in a group causes hiccup.
- Keeping the pipeline fed with instructions and data is very challenging
  - Memory access seem to take longer when measured in instruction cycles.
  - i-cache and d-cache size reduced to retain low latency at high frequency.
- Some pipeline hazards are more costly
  - Longer pipeline causes more cycles lost on reject/recycle
  - More cases cause reject/recycle rather than stall

Pipeline stalls and rejects

- Address Generation Interlock (AGI)
  - Waiting for the results of a previous instruction to compute an operand address
  - z10 has AGIs bypass that makes the results of Load Address and some Load instructions available before Putaway.
  - A group is stalled in the decode/issue unit until interlock is resolvable to avoid pipeline reject later
- Operand Store Compare (OSC)
  - Waiting to re-fetch a recently modified operand
  - A group is stalled in decode/issue unit based on inspection of i-text to avoid pipeline reject if OSC is encountered on reject/recycles part of pipeline

Pipeline stalls and rejects

- Instruction Fetch Interlock (IFI)
  - reloading instructions as a result of stores into the instruction stream (actually anywhere in the same cache line)
  - causes pipeline reject, clearing decoded instructions and refetching of instruction cache line (very costly)
- Branch Misprediction
  - branching (or not branching) in a way other than the processor has guessed.
  - z10 has complex branch prediction logic
  - relative branches have a lower penalty for misprediction
  - untaken branches don't need to be predicted
  - "code straightening" is a good idea

Inhibition of Superscalar Grouping

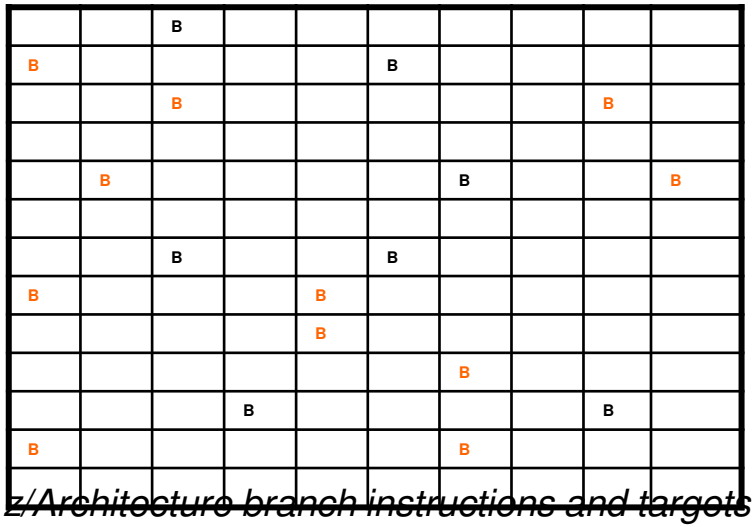
- Executing less than the optimal number of instructions simultaneously due to inter-instruction dependencies
  - requiring both the true and complimented value in a group causes a pipeline reject
  - There are a number of bypasses to eliminate dependencies that prevent grouping
    - Load Address AGI bypass
    - Load AGI bypass
    - Operand Forwarding

Branch Prediction

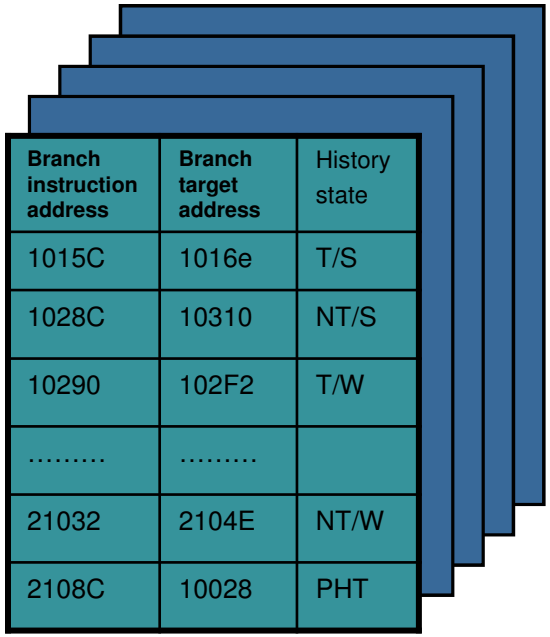
- The Branch Target Table remembers branches
  - BTB is indexed by part of the instruction address [halfword within 4K page]
  - Multiple states – *taken*, *strongly taken*, *not taken*, *strongly not taken*, *use PHT*
  - There is a Branch Pattern recording the last 9 branch directions (0/1)
  - A Pattern History Table is indexed by the Branch Pattern

Program Memory (halfwords)

Red "B"s are taken; Black "B"s are not taken



~~z/Architecture branch instructions and targets~~
can be on any halfword
BTB has a row for each halfword in a page

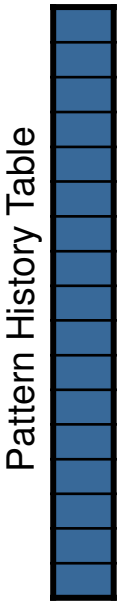


Branch instruction address	Branch target address	History state
1015C	1016e	T/S
1028C	10310	NT/S
10290	102F2	T/W
……	……	
21032	2104E	NT/W
2108C	10028	PHT

Branch Target Table
2048 x 5

Branch pattern

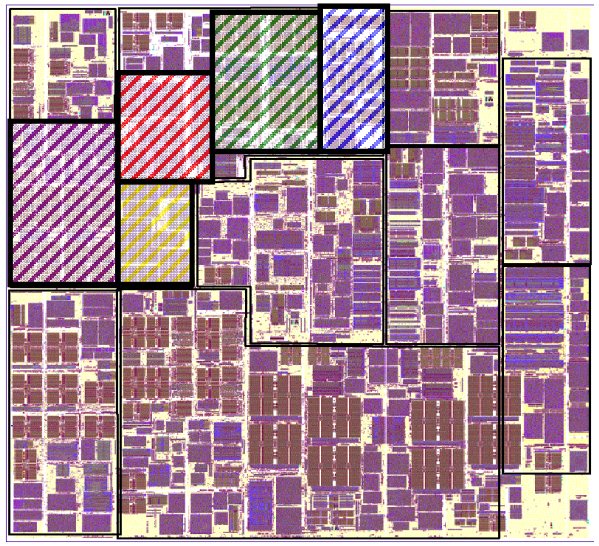
9-bits

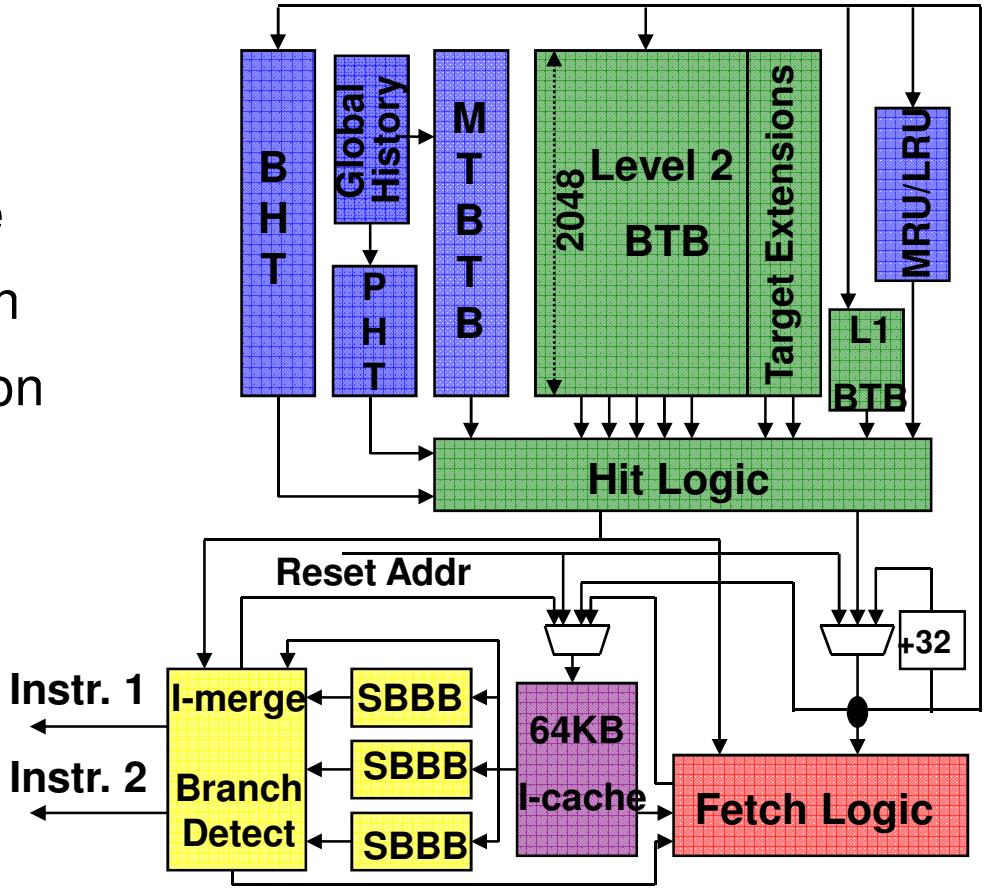


512 entries
2 bits wide
4 states

Branch Prediction

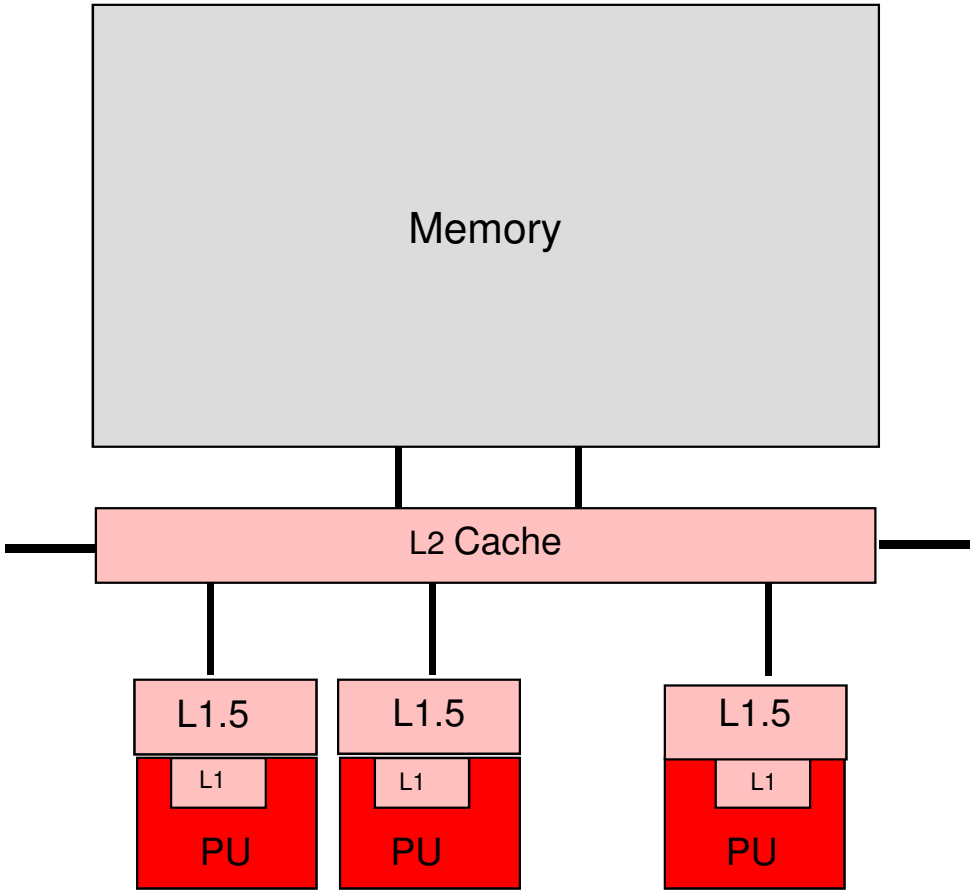
- **Multiple history-based prediction mechanisms**
  - 2 level Branch Target Buffer
  - Filtered Pattern History Table
  - Tagged multi-target prediction
  - Level 2 BTB data compression





z10 Cache Structure

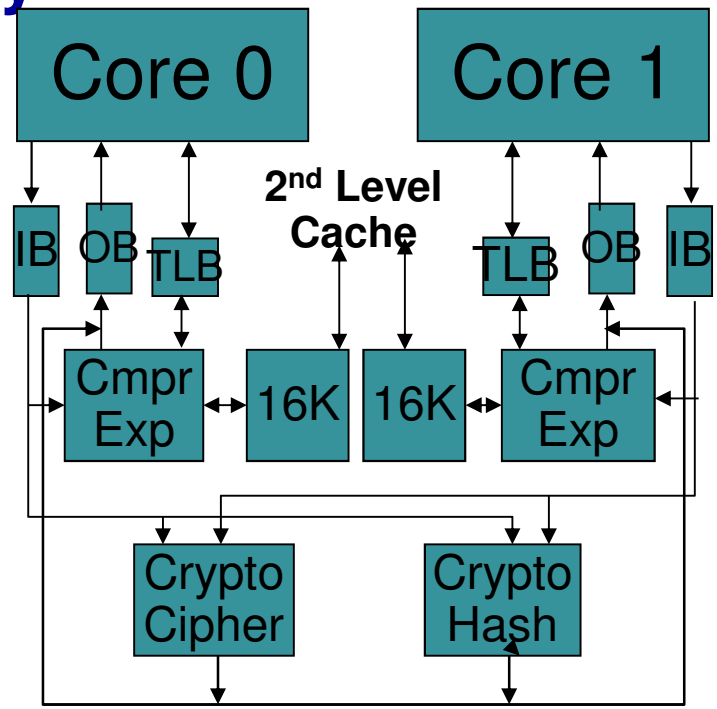
- Private Cache
  - L1 Instruction Cache
    - 64KB, 4-way set associative
  - L1 Data Cache
    - 128KB, 8-way set assocative
  - L1.5
    - 3MB, 12-way set associative
    - Inclusive of L1
- Shared Cache
  - L2
    - 48MB, 24-way set associative
    - Inclusive of L1 and L1.5
- Cache line size is 256 bytes
- Compare to z9
  - 256KB i-cache
  - 256KB d-cache
  - No L1.5
  - 40MB shared L2

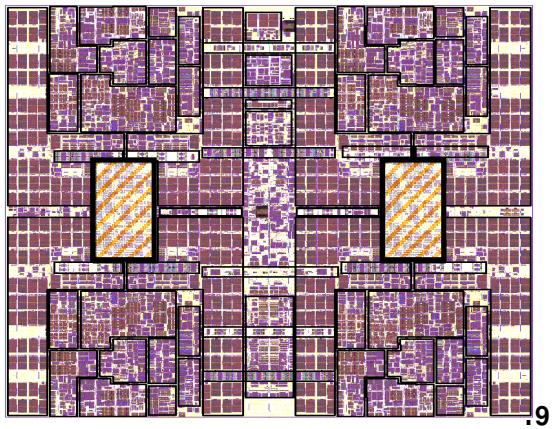


Up to 20 PUs sharing an L2

Compression and Cryptography Accelerator

- Accelerator unit shared by 2 cores
  - Independent compression engines
  - Shared cryptography engines
  - Co-operates with core millicode
  - Direct path into core store buffers
- Data compression engine
  - Static dictionary compression and expansion
  - Dictionary size up to 64KB (8K entries)
    - Local 16KB caches for dictionary data
  - Up to 8.8 GB/sec expansion
  - Up to 240 MB/sec compression
- Cryptography engine
  - DES (DEA, TDEA2, TDEA3)
  - SHA-1 (160 bit)
  - SHA-2 (256, 384, 512 bit)
  - AES (128, 192, 256 bit)
  - 290-960 MB/sec bulk encryption rate





z10 TLB2 and Large Pages

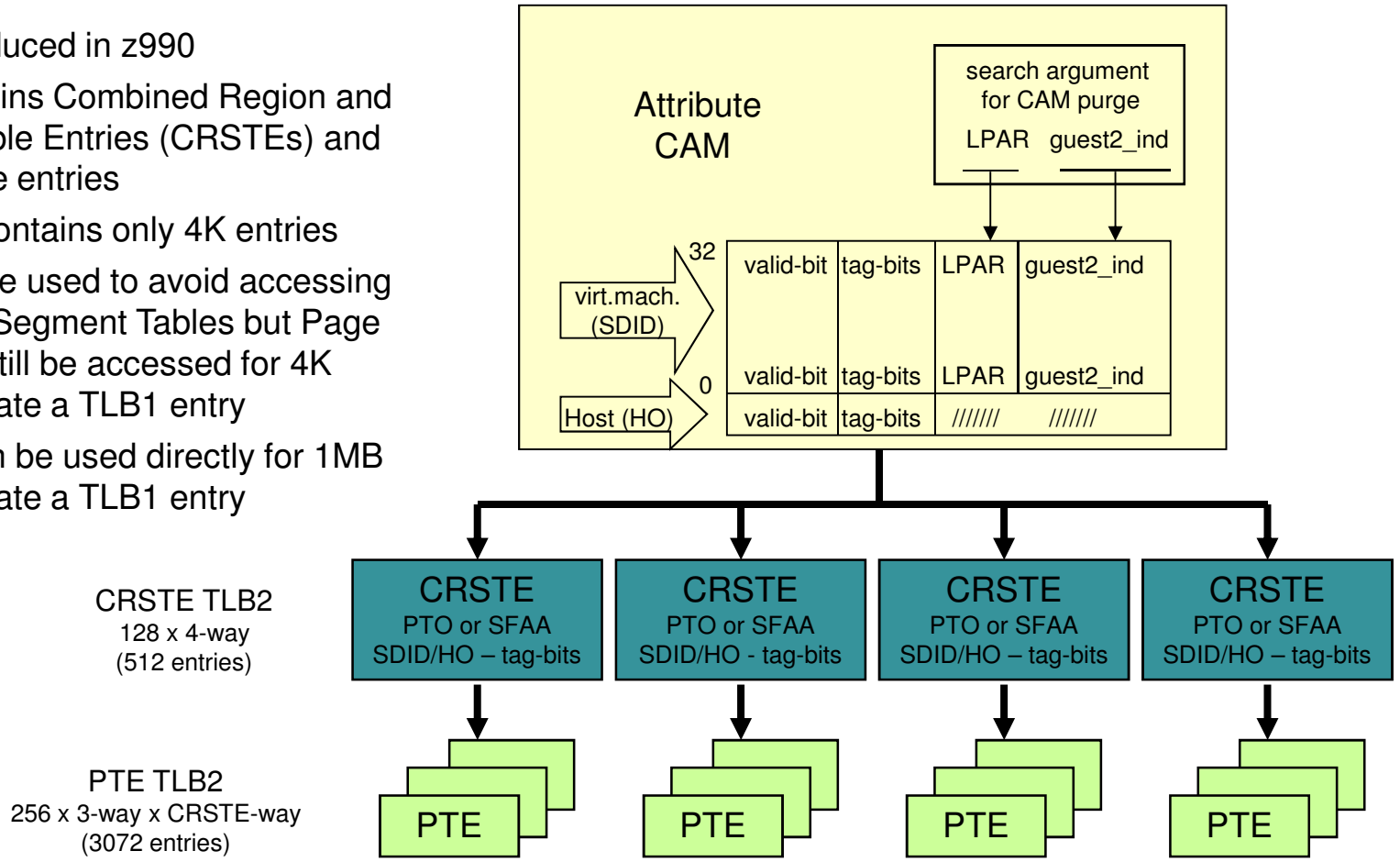
–TLB2 introduced in z990

–TLB2 contains Combined Region and Segment Table Entries (CRSTEs) and 4K pagetable entries

–TLB1 still contains only 4K entries

–CRSTEs are used to avoid accessing Region and Segment Tables but Page Table must still be accessed for 4K pages to create a TLB1 entry

–CRSTE can be used directly for 1MB pages to create a TLB1 entry



CRSTE TLB2
128 x 4-way
(512 entries)

PTE TLB2
256 x 3-way x CRSTE-way
(3072 entries)

TLB1 misses on Large Pages that hit in TLB2 can be resolved without accessing a page table entry

New Instructions on z10

- **Compare and Branch type**
  - To help on condition code limitation
- **Compare and Trap**
  - null pointer checks
- **Some new relative instructions**
  - Load Relative and Store Relative and “execute” relative
- **Immediate Instructions**
  - Move Immediate and compare immediate (16, 32, 64 bits)
  - Add Immediate (arithmetic and logical)
- **Fill necessary holes in latest architecture**
  - Some Multiply Immediate, some Multiply long displacement
- **Powerful bit manipulation instructions**
  - **Rotate Then (AND, OR, XOR, INSERT) Bits**