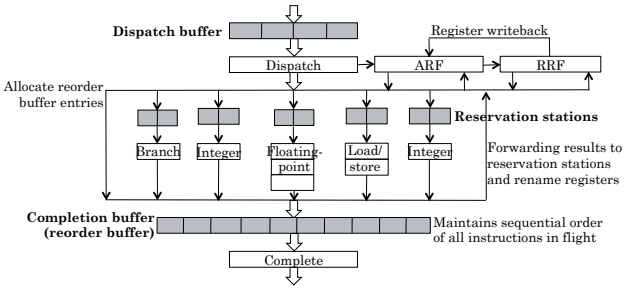
CS4100: Computer System Design

Dynamic Execution Core

Madhu Mutyam
PACE Laboratory
Department of Computer Science and Engineering
Indian Institute of Technology Madras

Oct 15-16, 2015

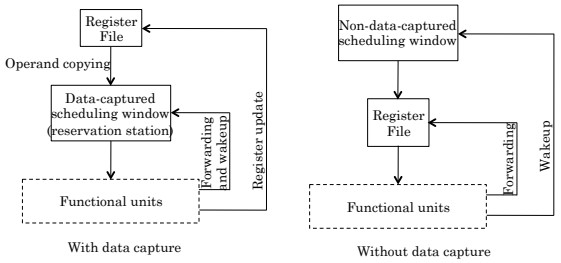
Dynamic Execution Core



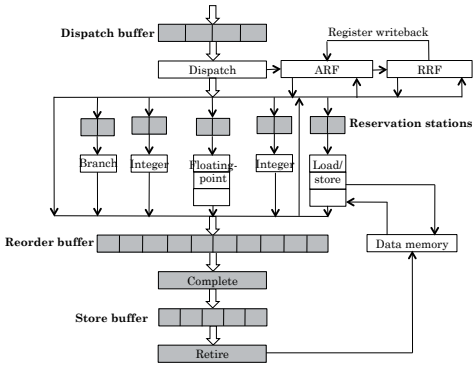
- *Instruction dispatching* – register renaming; Allocating RS and ROB entries; Advancing from dispatch buffer to RS
- *Instruction execution* – issuing ready instructions; executing the issued instructions; forwarding the results
- *Instruction completion*

Dynamic Instruction Scheduler

- Includes the instruction window (RS + ROB) and its associated instruction wake-up and select logic

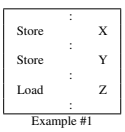


Processing of Load/Store Instructions



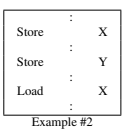
Ordering of Memory Accesses

- A memory data dependence exists between two load/store instructions if they both reference the same memory location
- Option #1: Execute all loads/store instructions in program order to enforce memory data dependences
- Stores must be executed in program order
  - Preserve the sequential state of the memory to recover from exceptions



Example #1

Load can bypass earlier *Stores*

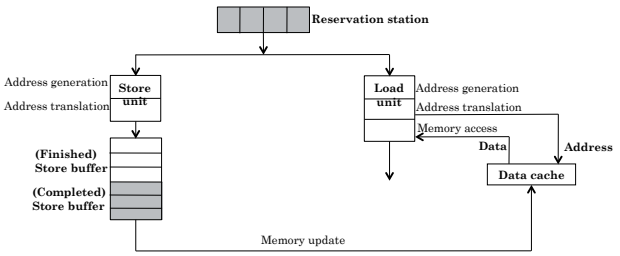


Example #2

Store can forward data to a later *Load*

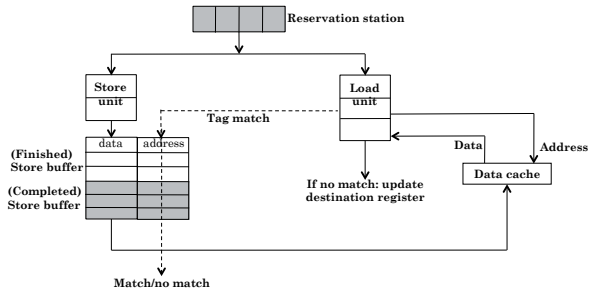
- Option #2: Out-of-order execution of loads is the primary source of performance gain
  - Make sure that RAW dependences are not violated

Load/Store Unit Organization



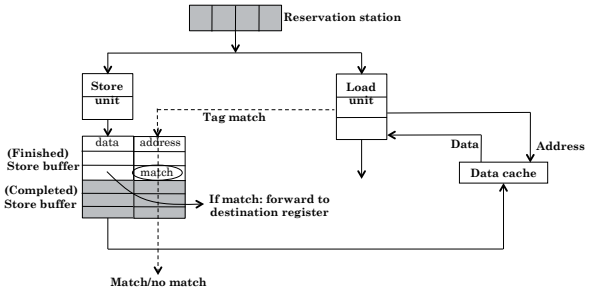
Assumption: Load/store instructions are issued from the shared reservation station in program order

Load Bypassing



Assumption: Load/store instructions are issued from the shared reservation station in program order

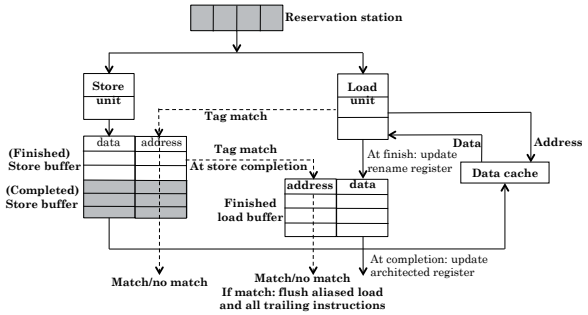
Load Forwarding



Assumption: Load/store instructions are issued from the shared reservation station in program order

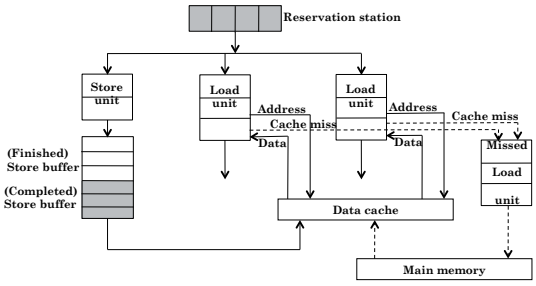
- Added complexity to the store buffer
  - Full length address bits
  - Priority encoding to identify the latest store
  - Additional read port

Fully O-o-O Issuing and Executing of Load and Store Instructions



- Exploit dependence prediction to determine whether to speculatively issue a load or not

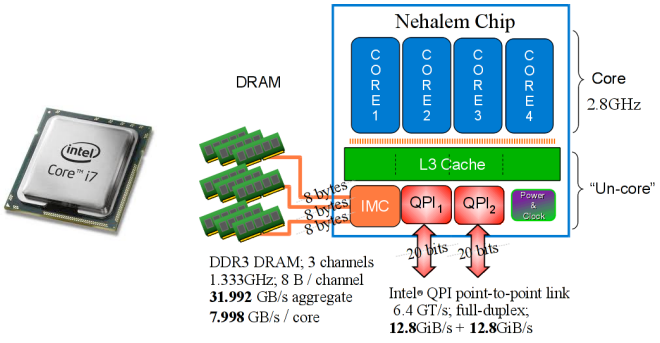
Dual-Ported and Non-Blocking Data Cache



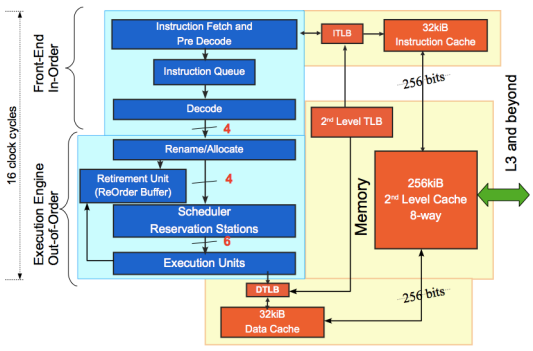
- Objective is to increase the memory bandwidth and reduce the memory latency

Nehalem Architecture[a]

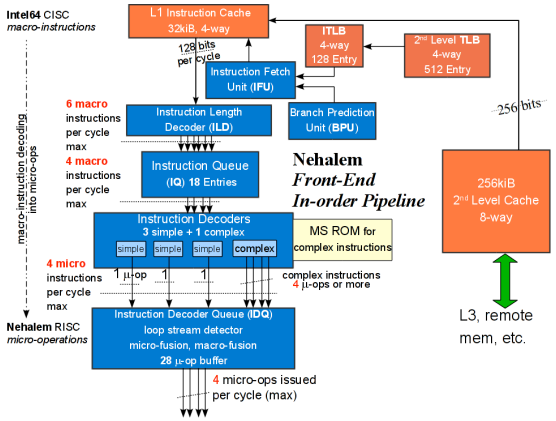
[a] M.E. Thomadikis. The Architecture of the Nehalem Processor and Nehalem-EP SMP Platforms.



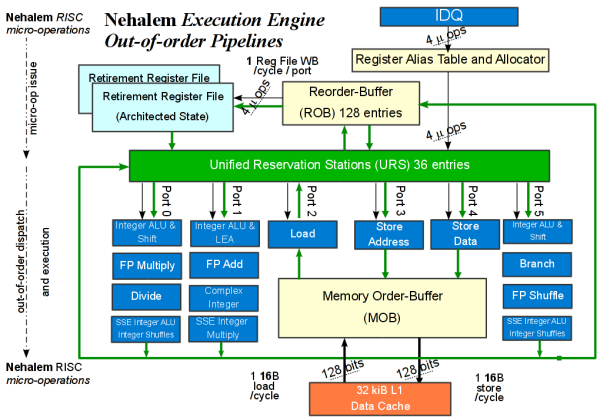
Nehalem Core Pipeline



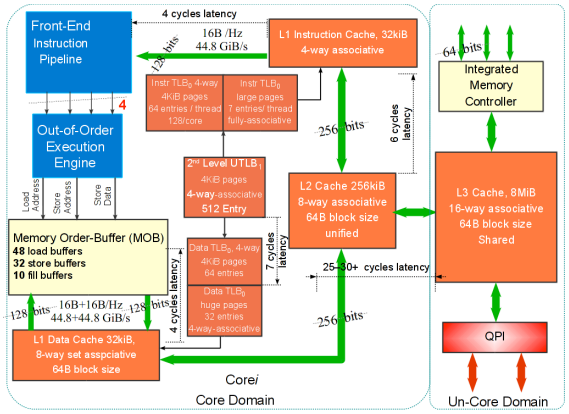
Front-End Pipeline



O-o-O Execution Engine



Memory Hierarchy



Thank You