Out-of-Order-Execution (OOOE)

CS441 Caleb Hellickson

What is it?

Out-of-Order-Execution (OOOE):

Maximizing the use of instruction cycles, by executing instructions on the availability of input data, rather than how they are presented in the program.

Why do I care?

Dependencies suck.

OOOE eliminates the following dependency issues:

- RAW
  - Dependent instruction put into reservation station
- WAR and WAW
  - Register renaming

History

- Idea was born out of research from data flow in 1970's.
- James E. Smith and A.R. Pleszkun published paper (1985)
  - *Implementing Precise Interrupts in Pipelined Processors*
- First Machine
  - CDC 6600 (1964)
    - Scoreboard to resolve conflicts
    - Not quite there yet

History cont.

- IBM 360/91 (1966)
  - Introduced Tomasulo's Algorithm
- IBM Power1 (1990)
  - OOOE Limited to floating point
- IBM PowerPC 601
- Fujitsu SPARC64
- Intel Pentium Pro
- Intel Atom

Robert Tomasulo's Algorithm

1. Get instruction
2. Rename it
3. Put it into reservation station
   a. If not enough reservation stations "stall"
   b. Watch common data bus for tags of source instructions and when ready grab it.
4. Dispatch instruction
5. After instruction finishes in FU (Executing)
   a. Arbitrate for CDB
   b. Put tagged value in CDB
   c. Connect Register to CDB and update if match

Tomasulo's Algorithm cont.

6. Reclaim and rename tag

How it works

Idea: Move dependent instructions of the way of independent ones by putting them into reservation stations and monitoring their state.

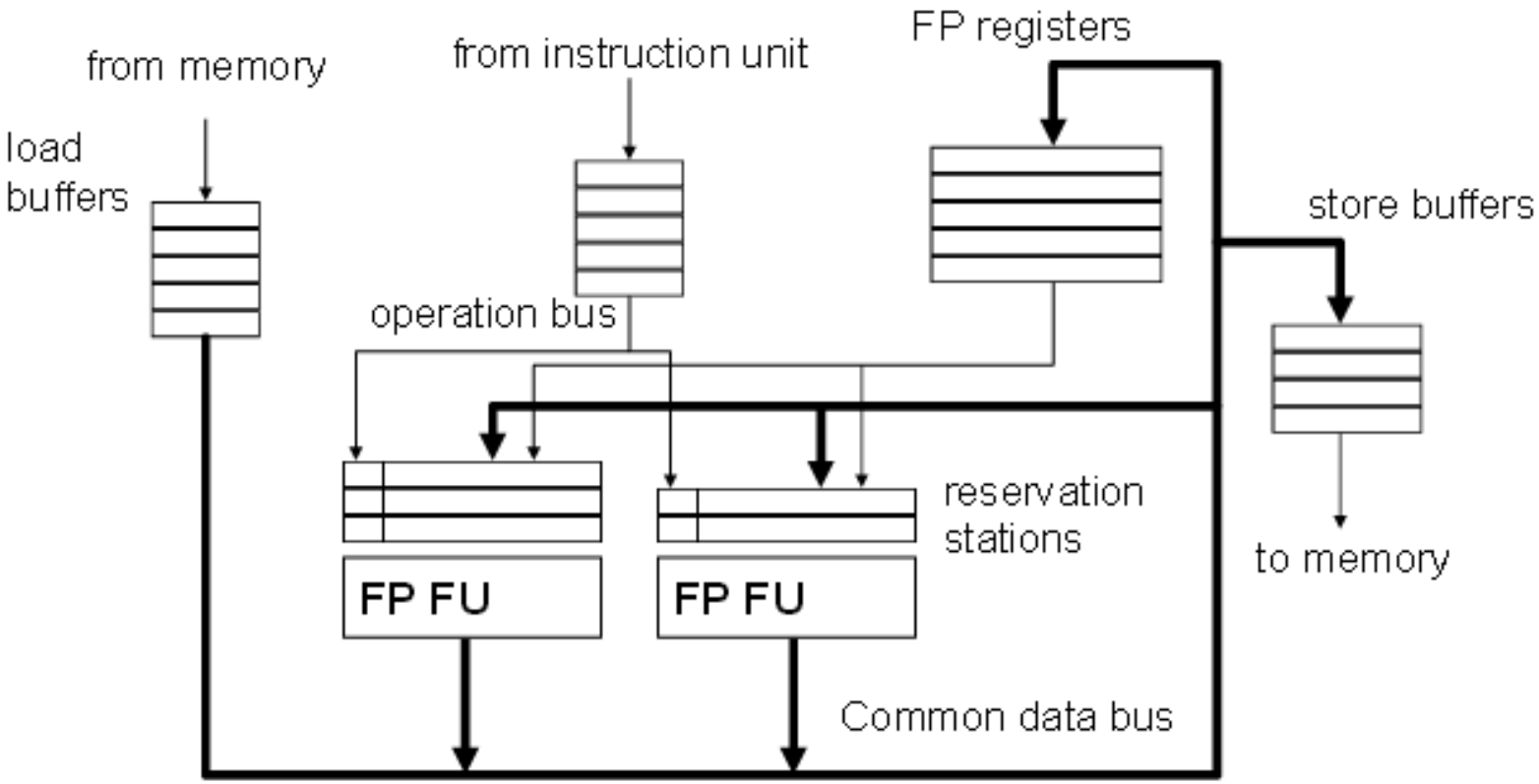
When all sources are ready, dispatch

Benefit: Allows independent instructions to execute and compute in the presence of a long latency operation

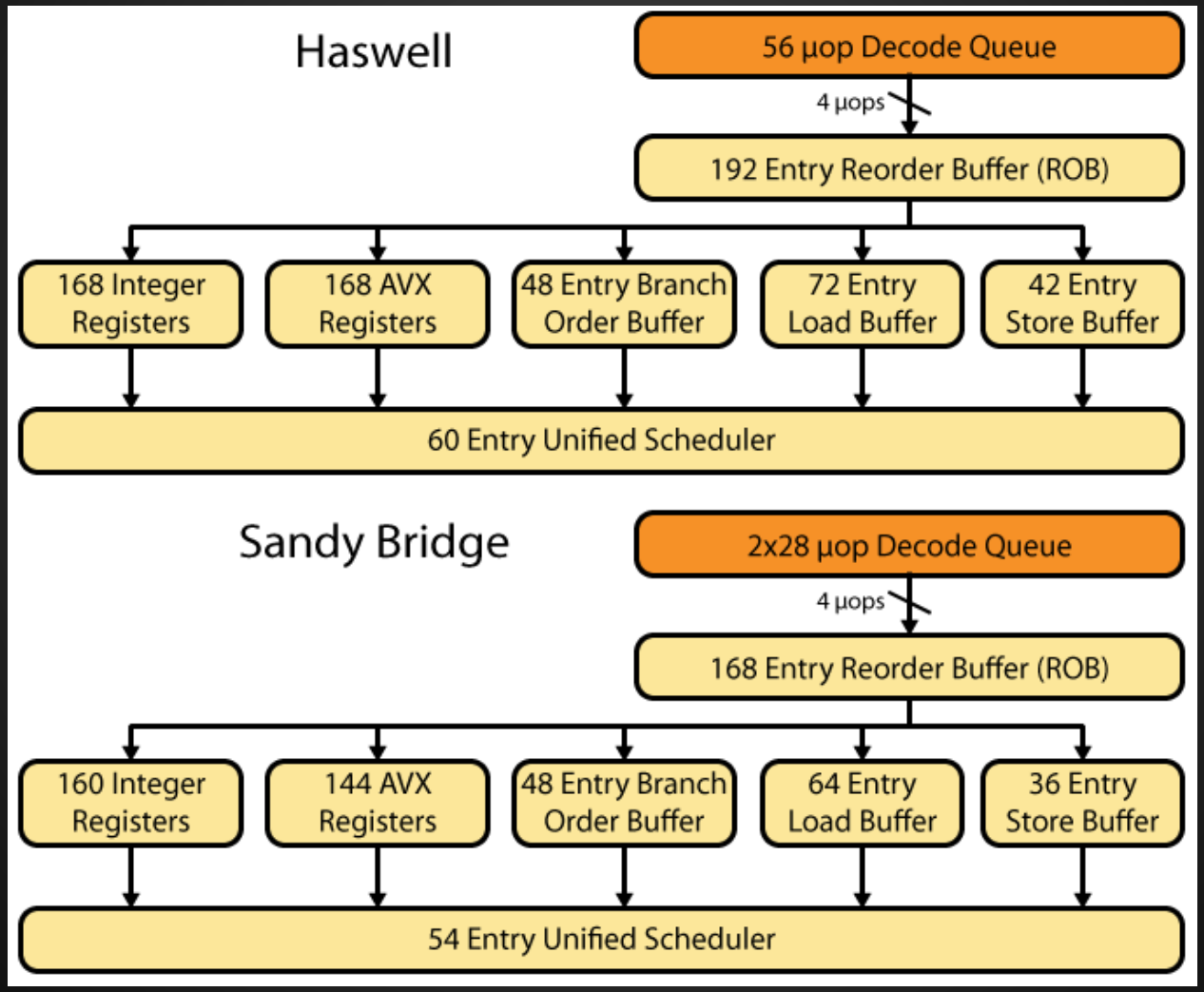
How it works cont

- Link consumer of value to producer
  - Register renaming: Associate “tag” with each data value.
- Buffer instructions until ready to execute
  - Put into reservation station after renaming
  - Broadcast “tag” when value produced
  - Instructions compare their “source tag” to broadcast “tag”
    - If they match source value is ready
- When all sources are ready, dispatch
  - Instruction wakes up
  - If multiple instructions, use one per FU

What's it look like?



Where It's Headed



Where It's Headed

- Haswell
  - Same size uop (micro-op) decode queue
  - Wider (24) entry reorder buffer
  - More Integer (8) and AVX Registers (24)
  - Same size entry branch reorder buffer
  - More Entry (8) and Store Buffers (6)

Where It's Headed

How do we benefit from Haswell upgrade?:

- Better register renaming
  - Does not have to handle all move uops
    - Saves resources!

- Larger Entry Reorder Buffer
  - Increases out-of-order window by 15%
    - Scheduling window over 300 operations since each fused uop (there are 4) gets an ROB entry

Where It's Headed

- More 256-bit AVX Registers
  - Accommodates new SIMD instructions
- Larger Load and Store Buffers
  - 72 loads and 42 stores in-flight

Questions?

References

Lectures (CMU):

http://www.youtube.com/watch?v=LU2W-YtyeEo

http://www.youtube.com/watch?v=LqENViWsThI

http://www.ece.cmu.edu/~ece740/f11/lib/exe/fetch.php?media=wiki:lectures:onur-740-fall11-lecture10-ooo-afterlecture.pdf

Background Info On OOOE in General:

http://en.wikipedia.org/wiki/Out-of-order_execution

Haswell Dynamic Scheduling Architecture:

http://www.realworldtech.com/haswell-cpu/3/