BEAST Lab
Pipelining & Branch Prediction in Modern CPUs

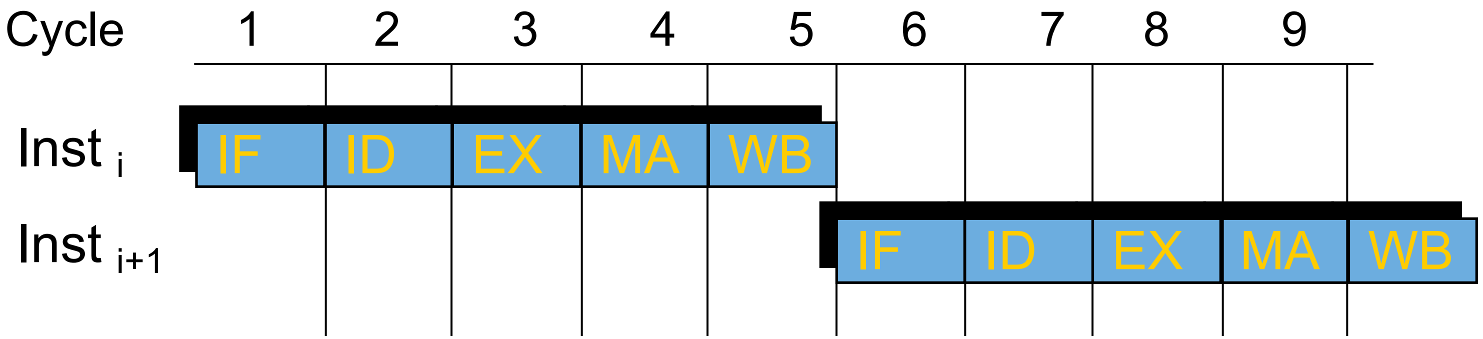
November 23, 2023 | Josef Weidendorfer / Sergej Breiter (LMU)

Instruction Execution in Stages (Example: MIPS)

- 5 different stages
  - IF: instruction fetch
  - ID: instruction decode & register fetch
  - EX: execute
    - run required operation in ALU
    - for load/store: calculate memory address
  - MA: memory access (optional)
  - WB: write back
    - result written back to register

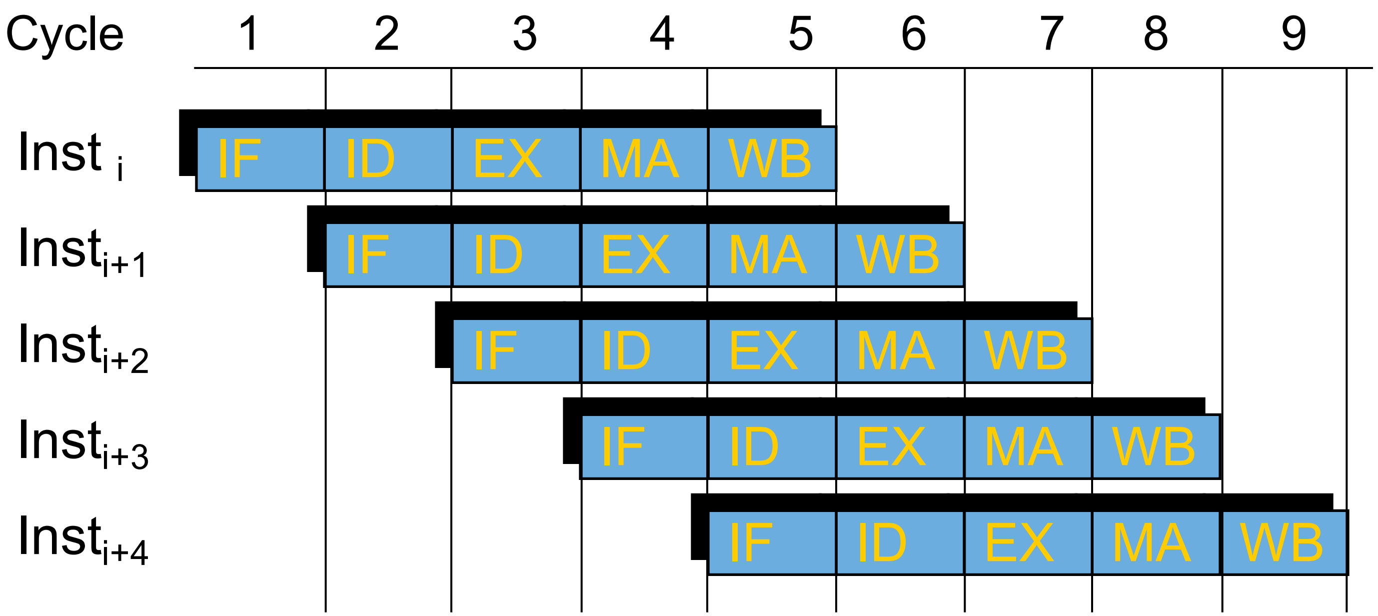
Timing without Pipelining

- instruction stream without jumps, 5 stages



- each stage executed by a different unit
  - each unit only used every 5th cycle
  - can be improved

Timing with Instruction Pipelining



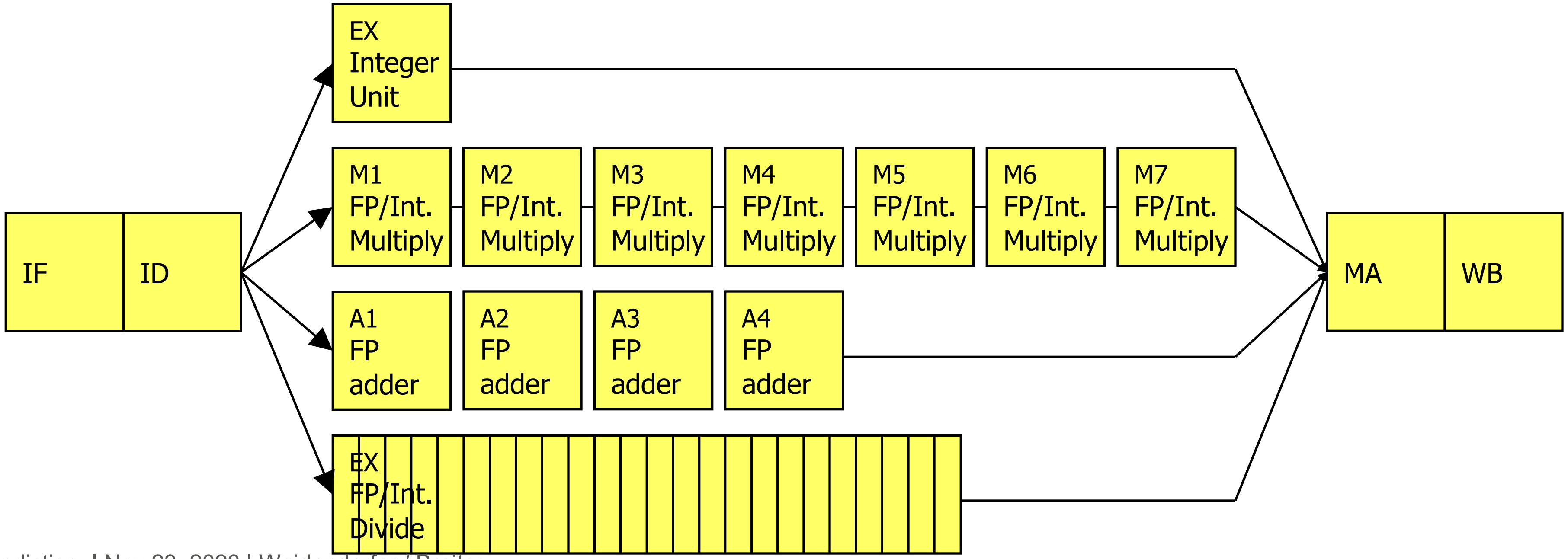
- similar to assembly lines in car industry
- pipeline with k stages, k=5
  - instruction latency still 5 cycles
  - throughput up to 1 instruction per cycle (CPI = 1)

Multiple Execution Units

Unit	Latency	Issue Latency
Integer ALU	1	1
FP Add	4	1
FP Multiply	7	1
FP Divide	25	25

reciprocal of maximal **throughput** (equal to latency if not pipelined)

Latency of an operation:
from point in time when inputs are available to when result is available



Examples (Latency / Issue Latency)

Intel Atom

- MOV r,r : 1 / 0.5
- PUSHF : ? / 14
- ADD r,r : 1 / 0.5
- ADC r,r : 1 / 1
- MUL r64 : 14 / 14
- IMUL r32 : 5 / 2
- FADD : 5 / 1
- FDIV : 71 / 71
- PMULUDQ: 5 / 2

Intel Haswell

may be eliminated

- MOV r,r : 0-1 / 0.25
- PUSHF : ? / 1
- ADD r,r : 1 / 0.25
- ADC r,r : 2 / 1
- MUL r64 : 3 / 1
- IMUL r32: 3 / 1
- FADD : 3 / 1
- FDIV : 10-24 / 8-18
- PMULUDQ: 5 / 1
- VFMA : 5 / 0.5
  (=16 SP FLOPS per cycle)

Source: http://www.agner.org/optimize/instruction_tables.pdf

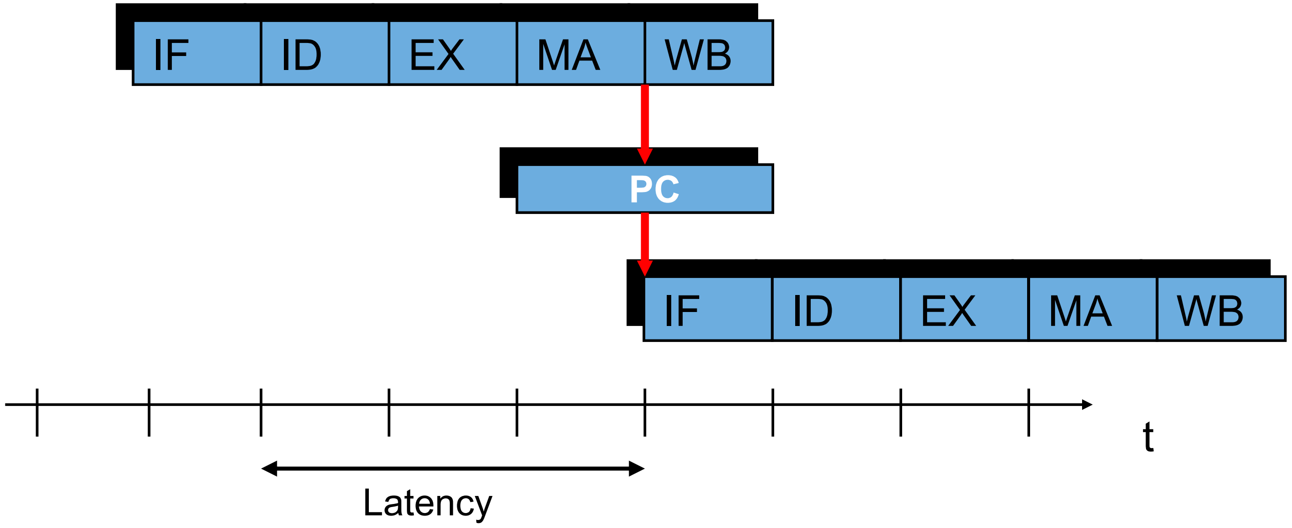
Pipelining: Examples

Number of pipeline stages

- Intel Pentium III: ~ 14
- Intel Pentium IV: ~ 24 (with FP up to 39)
- Intel Core / Atom: ~ 15
- ARM Cortex-A8: ~ 13
- ARM Cortex-M3: 3

Pipeline Conflict Avoidance for Control Conflicts

Example: conditional jump target available after EX, stored into PC in MA



Solutions:

- Wait 3 cycles, or
- Branch Prediction

Branch Prediction

- predict jump target in IF phase
- speculatively load instructions from predicted jump target
- stall if speculatively executed instructions would finish before jump target is evaluated
- after jump target evaluation
  - if wrong, throw away partly executed instructions
  - use prediction correctness to improve predictor

Predictor types

- static: always same prediction
  (fixed in HW, known to compiler / specified by compiler)
- dynamic: may change at runtime

Branch History Buffer (BHB)

- cache for prediction of conditional jumps
- enables prediction of jump target in ID phase
  - requires fast evaluation of jump target

Example

RISC, 32-bit

(PC >> 2) % 1024

Address-Tag	0
Address-Tag	0
Address-Tag	1
Address-Tag	1
inval	0
inval	0
Address-Tag	1
inval	0
inval	1
Address-Tag	1

1024 entries

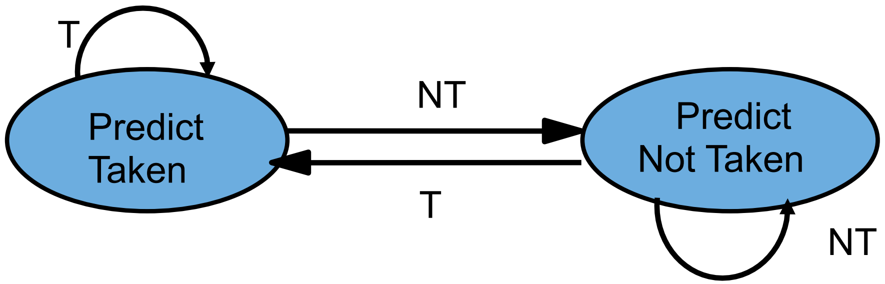
20 Bit

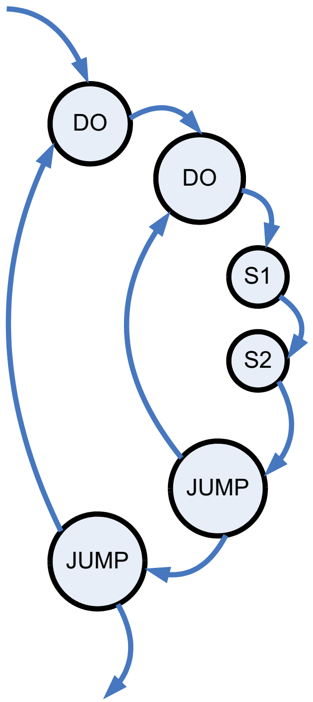
1-Bit Predictor

- 1: predict taken, 0: predict not taken
- if prediction wrong: invert

- assume 2 nested loops
  - predictor initialized with 1
  - fine in first iteration of outer loop, as long as inner loop is taken
  - on every next iteration of outer loop: 2 wrong predictions!

- better: 2-bit preditor





Branch Target Buffer (BTB)

- predict jump target address in IF phase
  - needed if jump target evaluation done late in pipeline
  - may use further predication bits: taken/not taken?

jump address (tag)	jump target	predication bits

  - updated on wrong prediction with actual jump address
  - if prediction correct: no stall at all !

Return Address Stack (RAS) Predictor

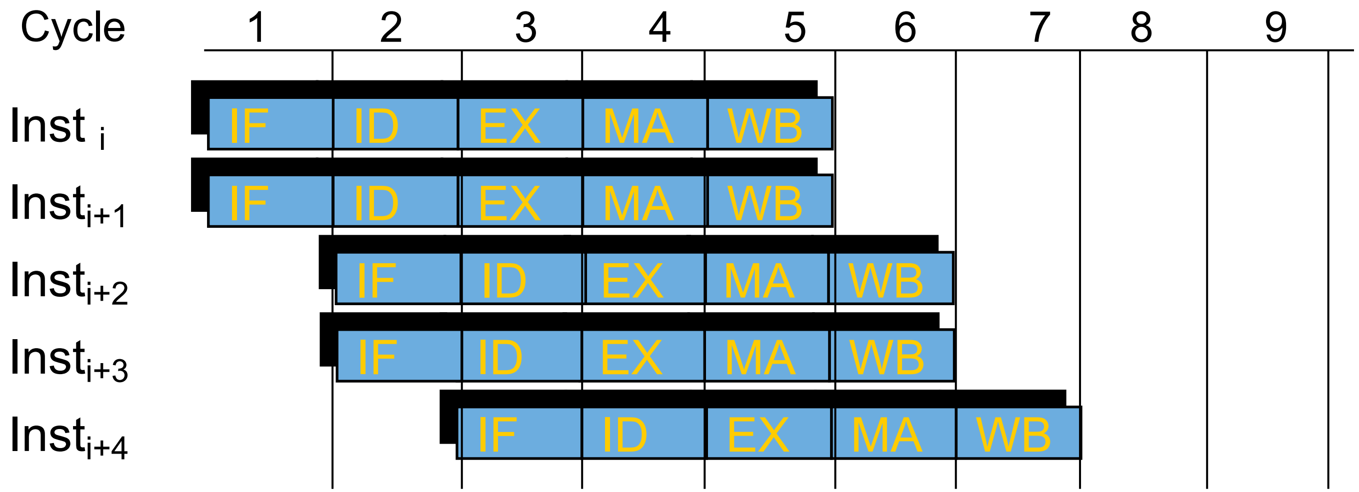
- functions usually called from multiple sites
- prediction of “return” (= indirect jump) often fails with a simple BTB

- use return address stack (RAS) for prediction
  - on a call (or branch updating link register): push return address
  - on return: predict top entry, pop from stack

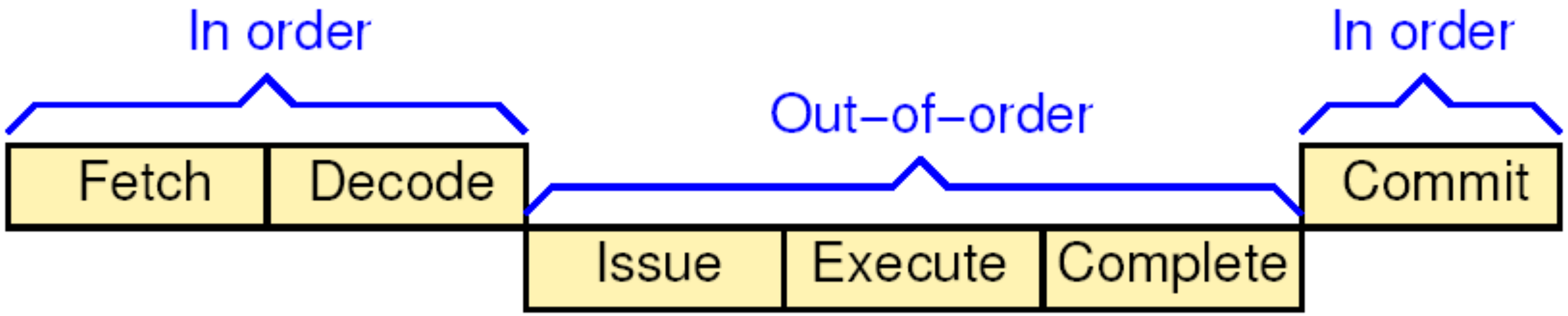
- mispredictions
  - code misuses call/return instructions
  - stack uses ring buffer: overriding old values

Superscalarity

- Duplication of pipeline stages
  - multiple instructions fetched, decoded, issued… **per cycle**



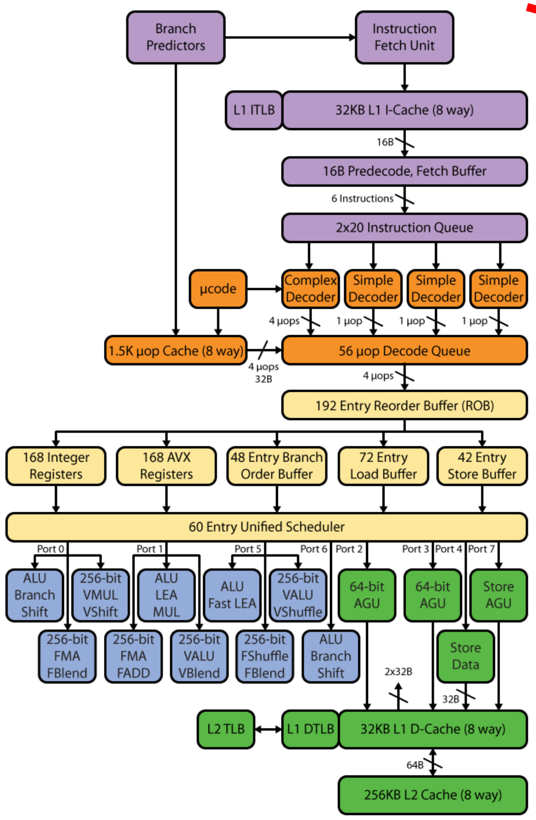
Superscalar Pipeline With Out-of-Order Execution



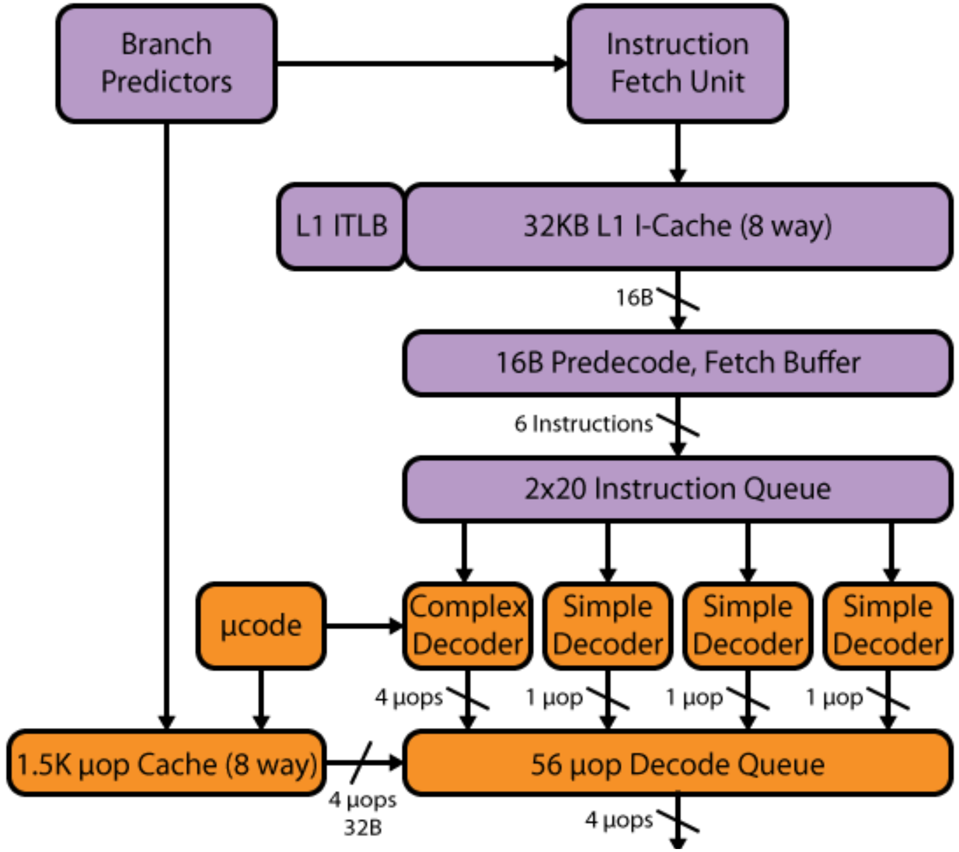
Commit: make result persistent

- If speculation wrong: result will be discarded

Example: Haswell (Intel Core 4th Gen)

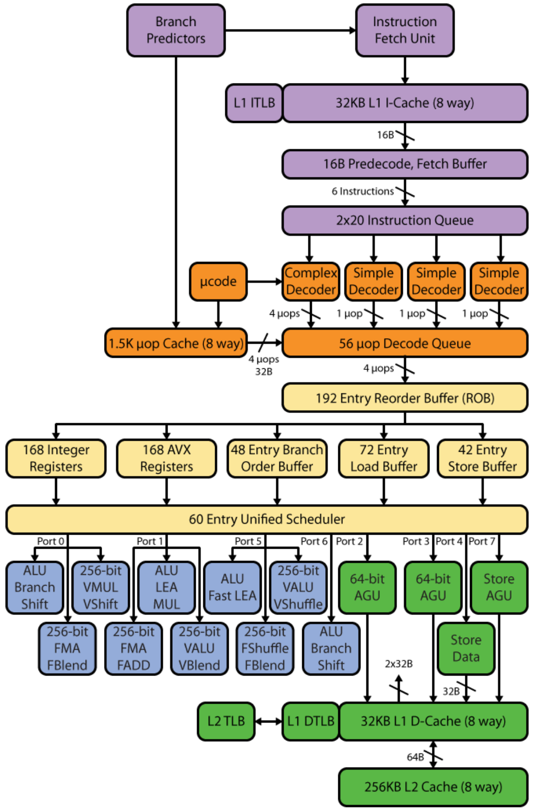


Frontend (IF/ID)

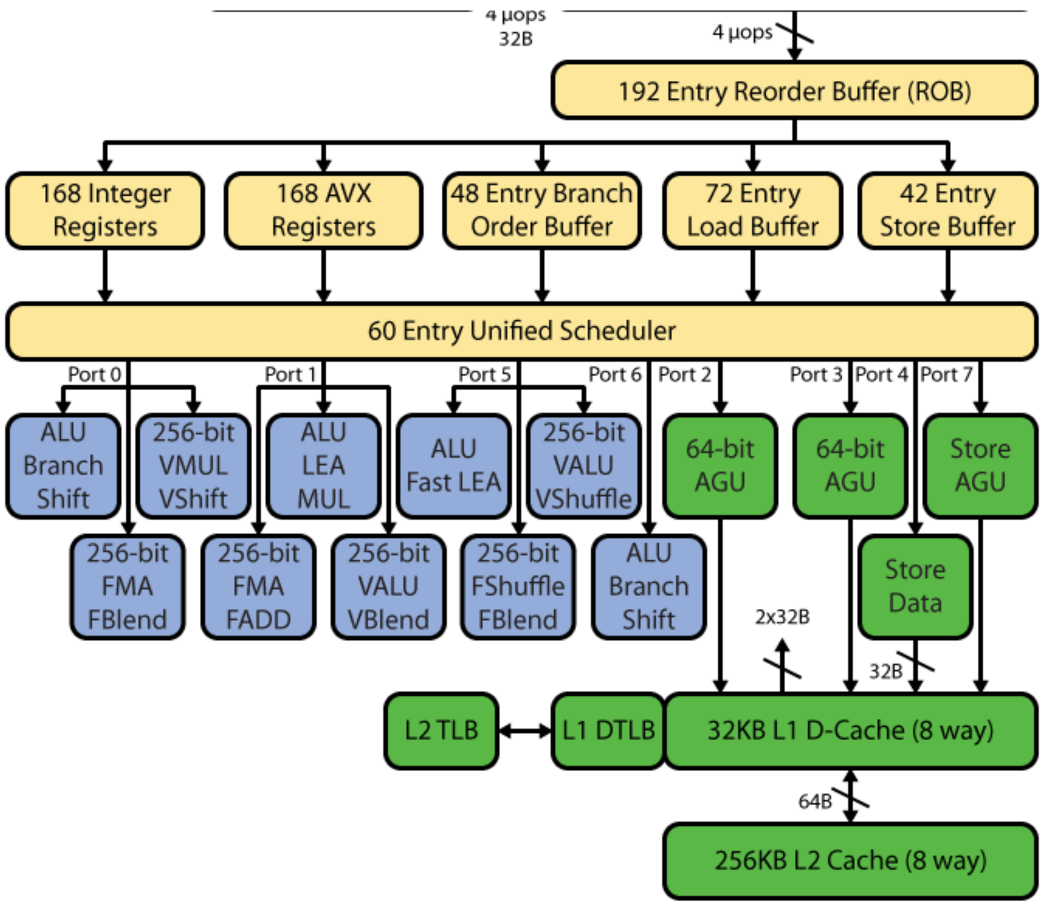


[David Kanter, RealWorldTech]

Example: Haswell (Intel Core 4th Gen)



Backend (Issue/Dispatch/Commit)



[David Kanter, RealWorldTech]