OOO IRL

What is the theoretical best CPI for our OOO CPU (with or without speculation)?

Multiple-issue

Allows for multiple instructions to be issued at the same time

Dynamically (by processor): superscalar

Statically (by compiler): Very Long Instruction Word (VLIW), EPIC

Issuing two instrs at once

lw t0 8(s0)

add t2, t0, t1

- What do the reservation stations/ROB look like?
- What does the hardware need to check in a *single cycle*?

Superscalar steps

1. Make sure there is room in the ROB (if speculative) and a reservation station for every instruction that *might* be in the next issue bundle (if necessary, bundles can be broken)
2. Analyze all dependences between instructions in bundle **(done in hardware in one cycle – HW grows quadratically in complexity w/ bundle size!)**
3. Update reservation stations info and ROB entries for all instructions in bundle and send them off to the FUs

ILP summary, so far

Deeper pipelining

- Resolve RAW hazards w/ stalling and forwarding
- Resolve control hazards w/ flushing OR branch prediction

OOO

- WAW and WAR hazards (resolved by register renaming/reservation stations)

OOO w/ speculation (in-order commit thanks to ROB)

- Allows CPU to make progress despite control hazards; requires branch prediction

Multiple issue

- Can work with pipelining, OOO, OOO + speculation
- Potentially better CPI

Bonus: VLIW

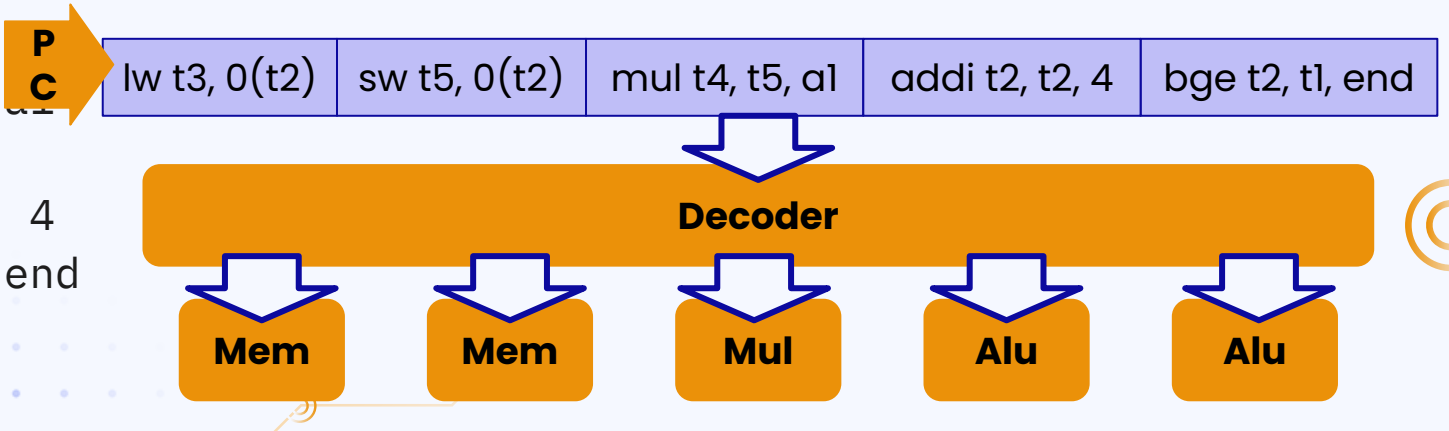
Compiler packs instructions into one **Very Long Instruction Word**

Early VLIW: no dependences between instructions, units operate in lockstep

Pros: No burden on CPU to track dependences

Cons: Dependent instructions(increased code size), compiler is uarch-dependent

```
lw t3, 0(t2)
mul t4, t5, a1
sw t5, 0(t2)
addi t2, t2, 4
bge t6, t1, end
```



Some real stuff

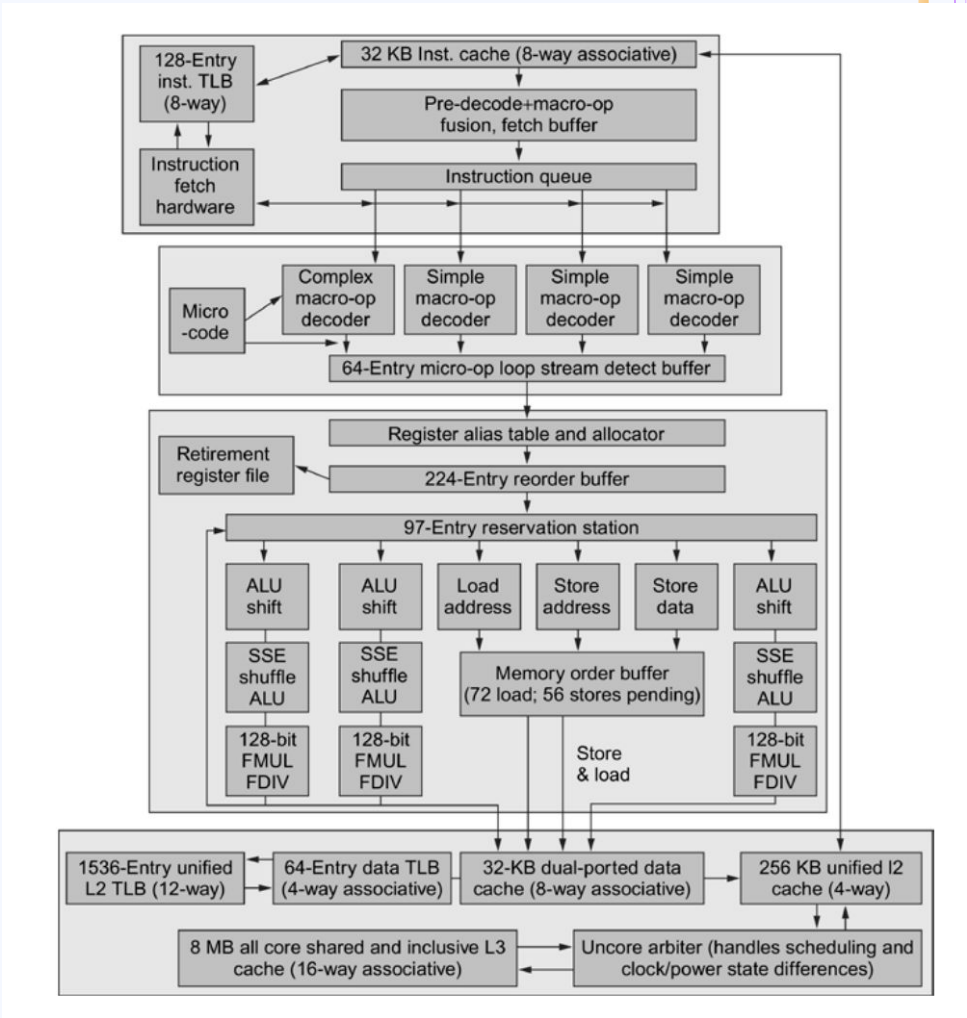
Intel Core i7 (H&P *7th ed.* fig. 3.38)

Not pictured (but Agner Fog talks about): µop cache, stack engine (for RAS)

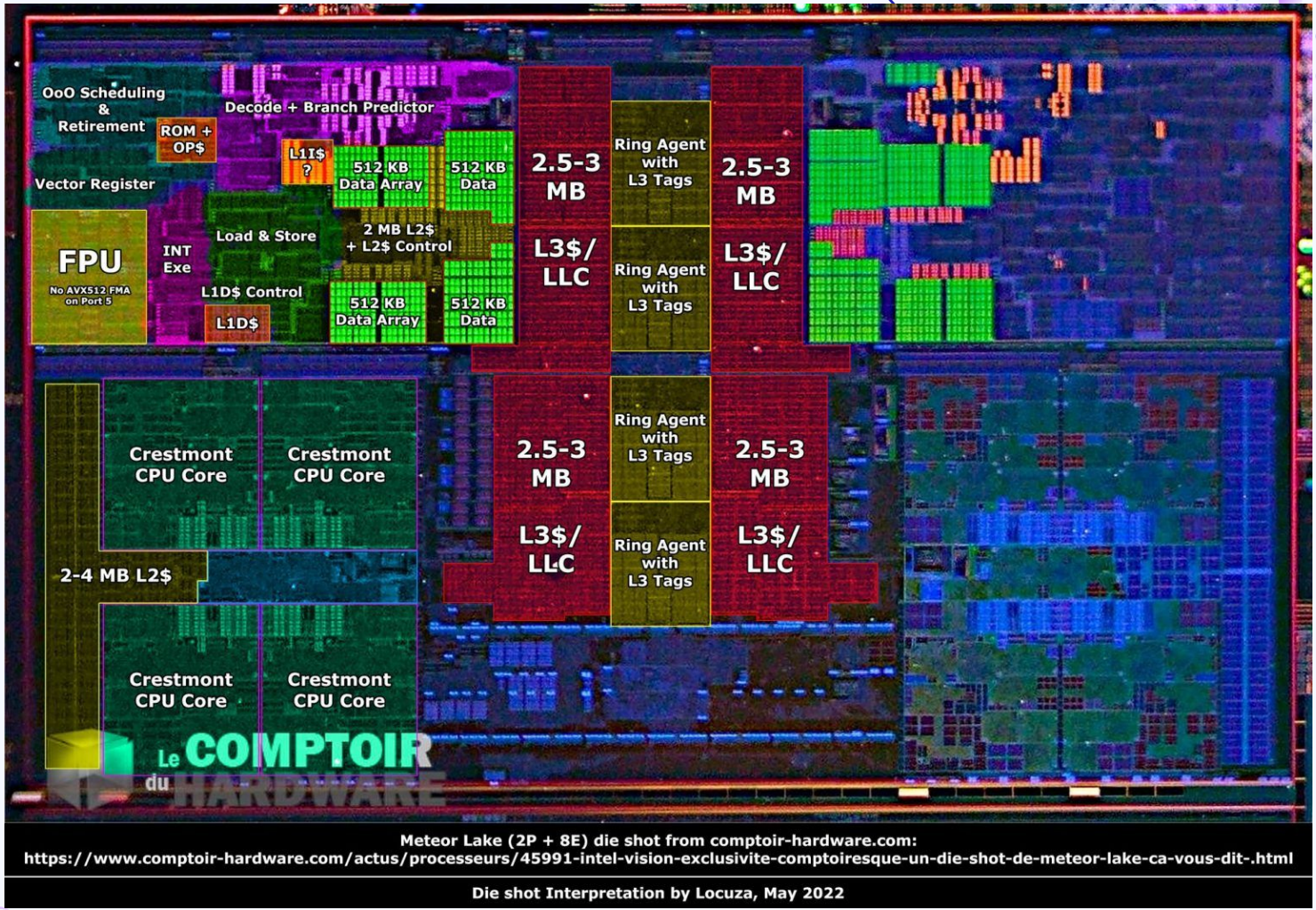
Macro-op fusion?

Loop stream detect?

SSE?



Intel “Meteor Lake”
source
NOTE: labeling not official; might not be fully accurate)



Meteor Lake (2P + 8E) die shot from comptoir-hardware.com:
https://www.comptoir-hardware.com/actus/processeurs/45991-intel-vision-exclusivite-comptoiresque-un-die-shot-de-meteor-lake-ca-vous-dit-.html

Die shot Interpretation by Locuza, May 2022

How is a micro-op cache indexed?

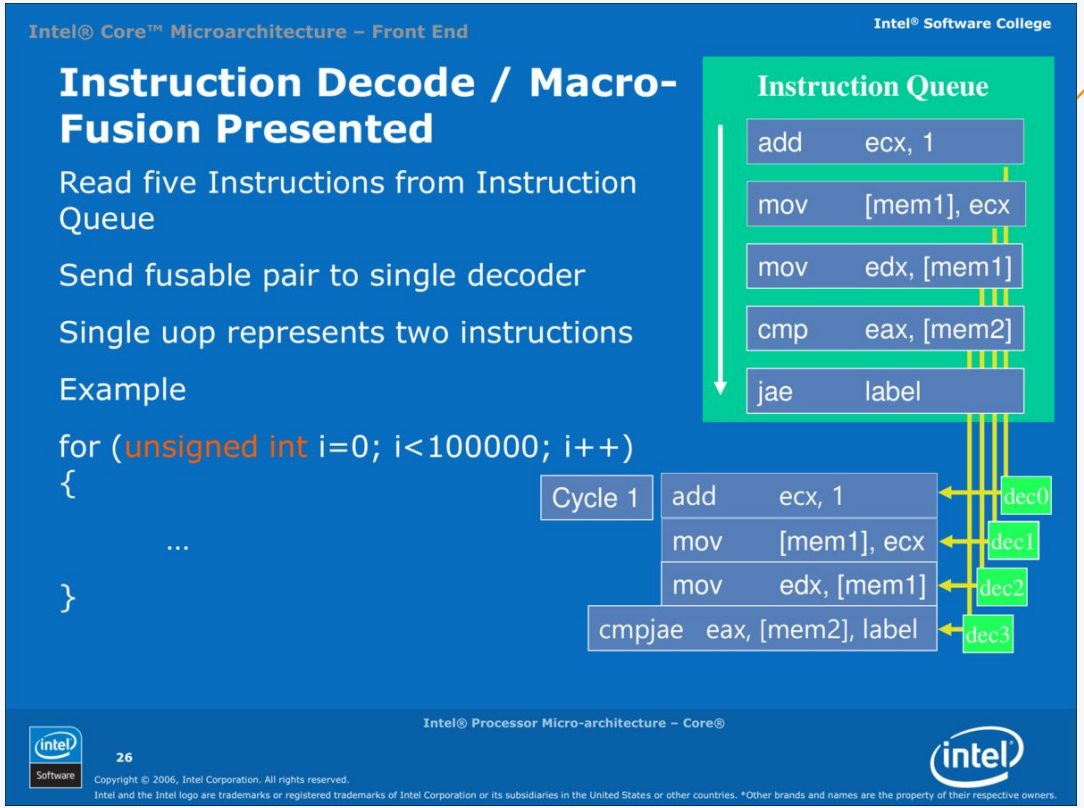
Macro-op fusion

MUL performs an XLEN-bit×XLEN-bit multiplication and places the lower XLEN bits in the destination register. MULH, MULHU, and MULHSU perform the same multiplication but return the upper XLEN bits of the full 2×XLEN-bit product, for signed×signed, unsigned×unsigned, and signed×unsigned multiplication respectively. If both the high and low bits of the same product are required, then the recommended code sequence is: MULH[[S]U] *rdh, rs1, rs2*; MUL *rdl, rs1, rs2* (source register specifiers must be in same order and *rdh* cannot be the same as *rs1* or *rs2*). Microarchitectures can then fuse these into a single multiply operation instead of performing two separate multiplies.

DIV and DIVU perform signed and unsigned integer division of XLEN bits by XLEN bits. REM and REMU provide the remainder of the corresponding division operation. If both the quotient and remainder are required from the same division, the recommended code sequence is: DIV[U] *rdq, rs1, rs2*; REM[U] *rdr, rs1, rs2* (*rdq* cannot be the same as *rs1* or *rs2*). Microarchitectures can then fuse these into a single divide operation instead of performing two separate divides.

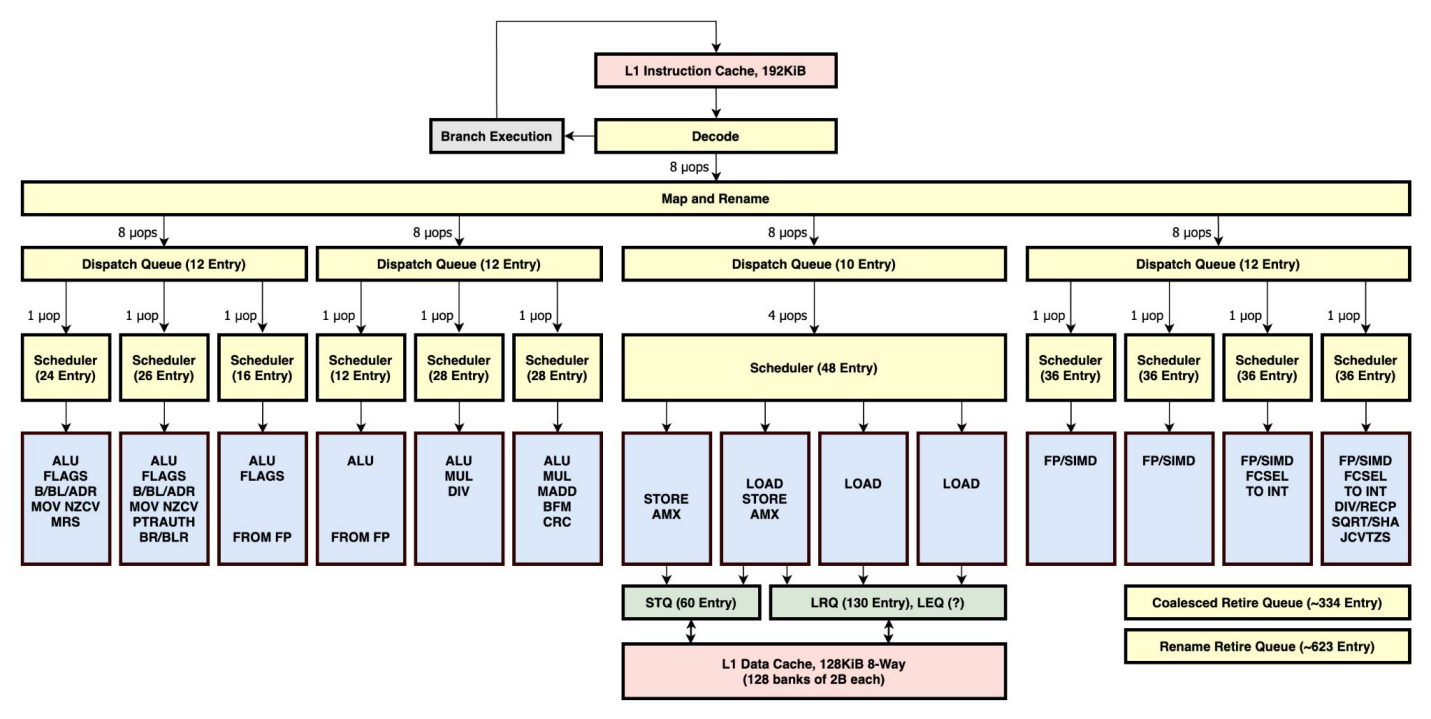
Macro-op fusion

source



Firestorm (Apple M1)

Source (NOTE: reverse-engineered: might not be fully accurate)



Arm Cortex SW Optimization Guide

Why is this a guide that's specific to a uarch (as opposed to an ISA)?

Some interesting observations (link)

- Issue width/dispatch constraints (p62)
- Recommendations for loads/stores (p63)
- (non) renaming of special registers (p65)
- Macro-op fusion (p68)