



# Multiple issue (superscalar)

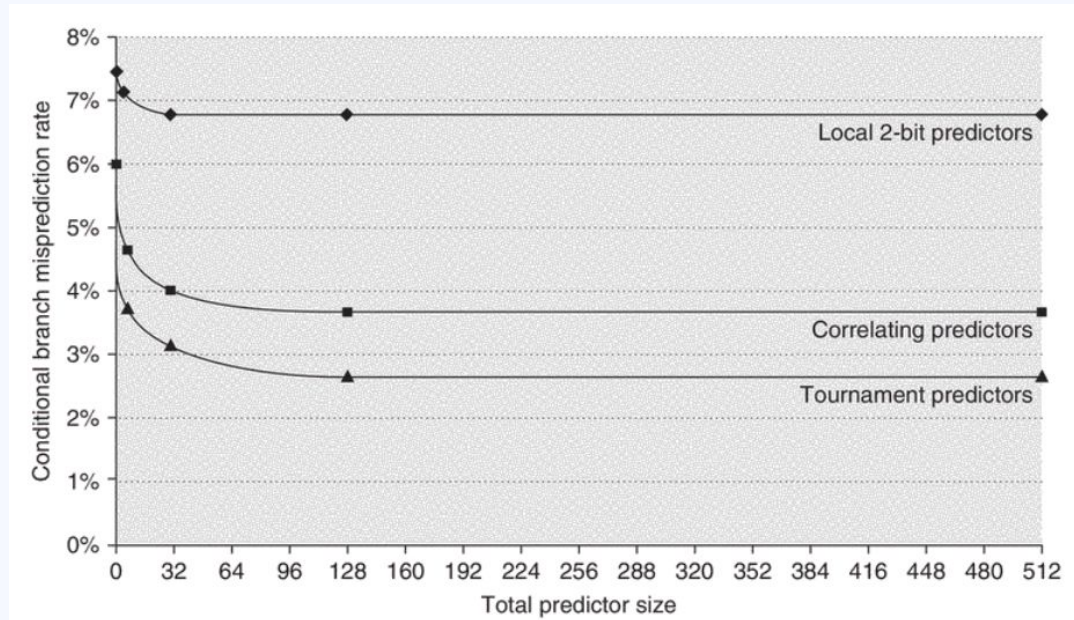


# Tournament branch predictors

Run local predictors and global predictor

Keep track of which one is doing better (using eg 2-bit predictor!) and use that one for each branch

H&P fig. 3.4





Even if we had perfect branch prediction, what would we need in order to make sure that speculative execution is fast?

# Branch target buffers

Cache for computing branch or jump target address (new PC)

Potentially faster to fetch next instruction

Common in modern systems; unlike branch delay slots

Multiple ways to set this up (see [Agner document](#))

Multiple levels

Different behavior for different types of branches/jumps



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

Multiple variations: in-order, OOO, OOO + speculative

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



Potentially how many instructions can we issue at once if we have the following FUs?

With what caveats?

- 1 load
- 1 store
- 2 integer ALUs
- 1 FP add/sub
- 1 FP mul/div

# 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: RAW hazards, control hazards

OOO

OOO w/ speculation

Multiple issue

# Bonus: VLIW

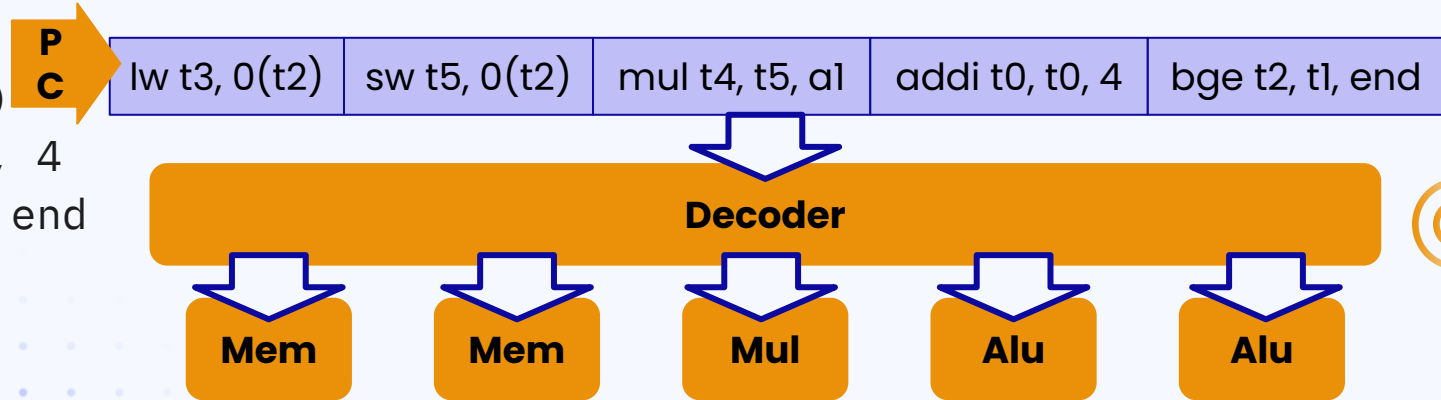
Compiler packs instructions into one **Very Long Instruction Word**

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

Pros:

Cons:

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

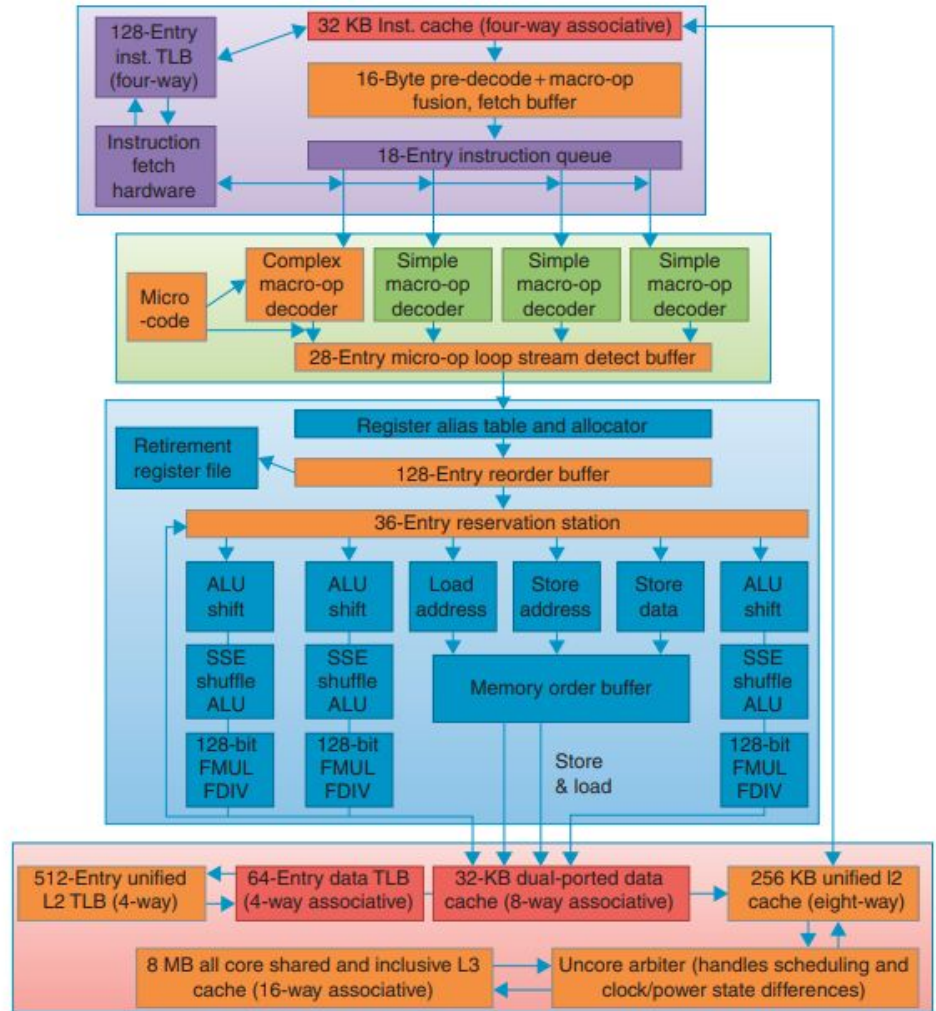




**Some real stuff**

# Intel Core i7 (H&P fig. 3.41)

Pre-decode??  
Complex macro-op  
decoder??  
Loop stream detect??



# Macro-op fusion

source

Intel® Core™ Microarchitecture – Front End Intel® Software College

## Instruction Decode / Macro-Fusion Presented

Read five Instructions from Instruction Queue

Send fusable pair to single decoder

Single uop represents two instructions

Example

```
for (unsigned int i=0; i<100000; i++)  
{  
    ...  
}
```

### Instruction Queue

- add ecx, 1
- mov [mem1], ecx
- mov edx, [mem1]
- cmp eax, [mem2]
- jae label

Cycle 1

add	ecx, 1	← dec0
mov	[mem1], ecx	← dec1
mov	edx, [mem1]	← dec2
cmpj	eax, [mem2], label	← dec3

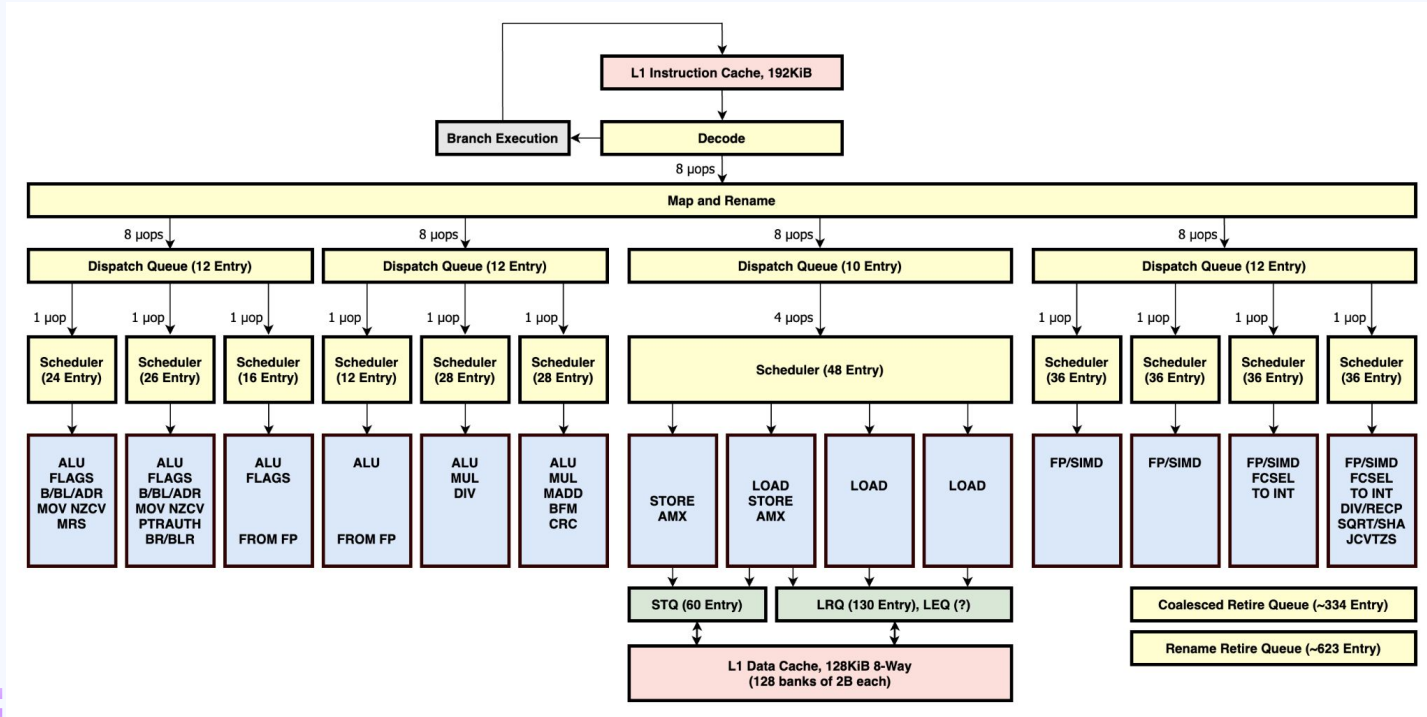
Intel® Processor Micro-architecture – Core®

26

Copyright © 2006, Intel Corporation. All rights reserved.  
Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States or other countries. \*Other brands and names are the property of their respective owners.

# 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)