

The top-left corner of the slide features a decorative graphic of circuit lines. It includes a horizontal blue line, a diagonal purple line, and several orange lines with circular nodes at their intersections, resembling a microchip or circuit board layout.

# **Multiple issue (superscalar)**

The bottom-right corner contains another decorative graphic. It features a grid of small blue dots, a diagonal line of blue dots, and several orange lines with circular nodes, similar to the top-left corner. There are also some blue geometric shapes and a small cluster of orange dots.

# From ISA lecture: micro-ops

Translation of complex machine instruction (macro-op) to multiple steps

Microarchitecture dependent (Intel doesn't provide documentation on this)

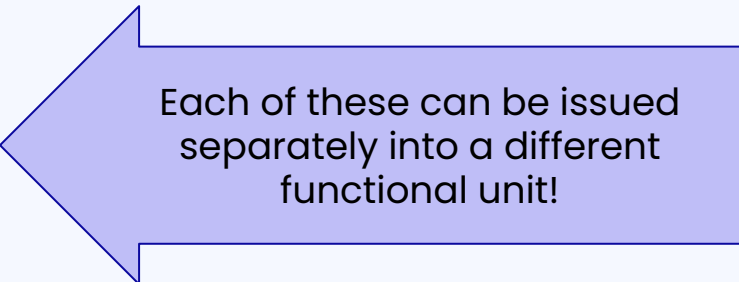
For example, `addq 8(%rdi) %rax` might be translated into:

add 8 to rdi

load that address from memory

add that value to rax

store the result in rax



Each of these can be issued separately into a different functional unit!

x86 processors have had a uop decoder since the Pentium Pro (1997)

Even RISC processors decode into uops – driven by design of FUs



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

We'll come back to this very soon



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

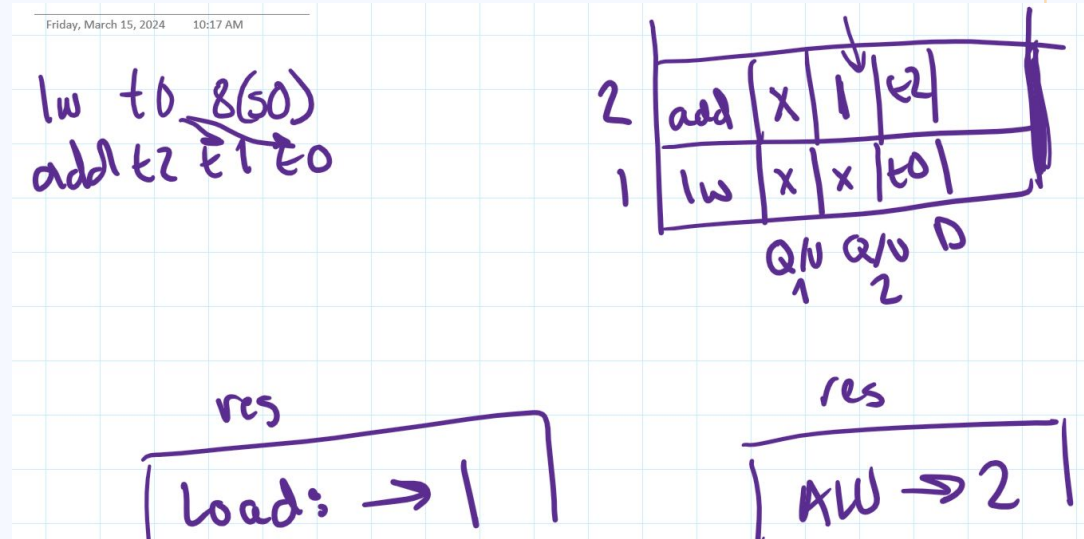
- 1 load
- 1 store
- 2 integer ALUs
- 1 FP add/sub
- 1 FP mul/div

# 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

# Example

l: lw t0, 0(s0)

addi t0, t0, 1

sw t0, 0(s0)

addi s0, s0, 8

bne t0, t1, l



l: lw t0, 0(s0)	I	E	E	W										
<u>addi</u> t0, t0, 1		I		E	W									
sw t0, 0(s0)			I	E		E								
<u>addi</u> s0, s0, 8				I	E	W								
bne t0, t1, 1					I		E							
l: lw t0, 0(s0)						I		E	E	W				
<u>addi</u> t0, t0, 1							I				E	W		
sw t0, 0(s0)								I	E				E	
<u>addi</u> s0, s0, 8									I	E	W			
bne t0, t1, 1										I			E	

				0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
l: lw t0, 0(s0)	I	E	E	W														
<u>addi</u> t0, t0, 1	I				E	W												
sw t0, 0(s0)		I	E				E											
<u>addi</u> s0, s0, 8		I	E	W														
bne t0, t1, 1			I				E											
l: lw t0, 0(s0)			I					E	E	W								
<u>addi</u> t0, t0, 1				I								E	W					
sw t0, 0(s0)				I					E						E			
<u>addi</u> s0, s0, 8					I				E	W								
bne t0, t1, 1					I										E			

w/o  
speculation

w/ speculation

l: lw t0, 0(s0)	I	E	E	W	C											
<u>addi</u> t0, t0, 1		I			E	W	C									
sw t0, 0(s0)			I	E				C								
<u>addi</u> s0, s0, 8				I	E	W			C							
bne t0, t1, 1					I		E			C						
l: lw t0, 0(s0)						I	E	E	W		C					
<u>addi</u> t0, t0, 1							I			E	W	C				
sw t0, 0(s0)								I	E				C			
<u>addi</u> s0, s0, 8									I	E	W			C		
bne t0, t1, 1										I		E			C	

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
l: lw t0, 0(s0)	I	E	E	W	C										
<u>addi</u> t0, t0, 1	I				E	W	C								
sw t0, 0(s0)		I	E				C								
<u>addi</u> s0, s0, 8		I	E	W				C							
bne t0, t1, 1			I				E	C							
l: lw t0, 0(s0)				I	E		E	W	C						
<u>addi</u> t0, t0, 1				I					E	W	C				
sw t0, 0(s0)					I	E					C				
<u>addi</u> s0, s0, 8					I	E	W					C			
bne t0, t1, 1						I					E	C			

# ILP summary, so far

Deeper pipelining: RAW hazards, control hazards

Strategies: stalling, flushing, forwarding, branch prediction, compiler reordering instrs

OOO

CPU reorders instrs (compiler doesn't need to know about uarch)

Difficulties: WAW, WAR for stores, branches

OOO w/ speculation

Commit in order! Use branch prediction

Multiple issue

Potential gains in CPI

# Arm Cortex SW Optimization Guide

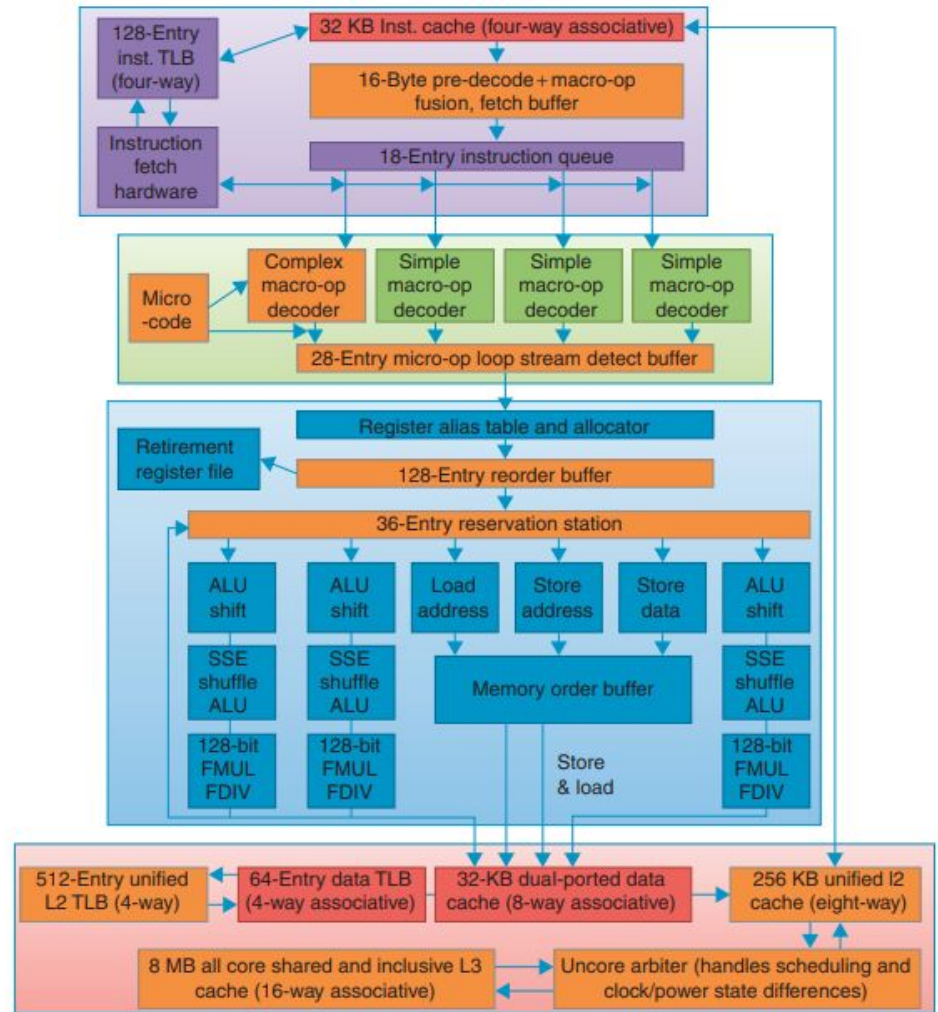
***Why is this a guide that's specific to a uarch (as opposed to an ISA)?***

Some interesting observations ([link](#))

- Issue width
- Recommendations for loads/stores
- (non) renaming of special registers
- Macro-op fusion

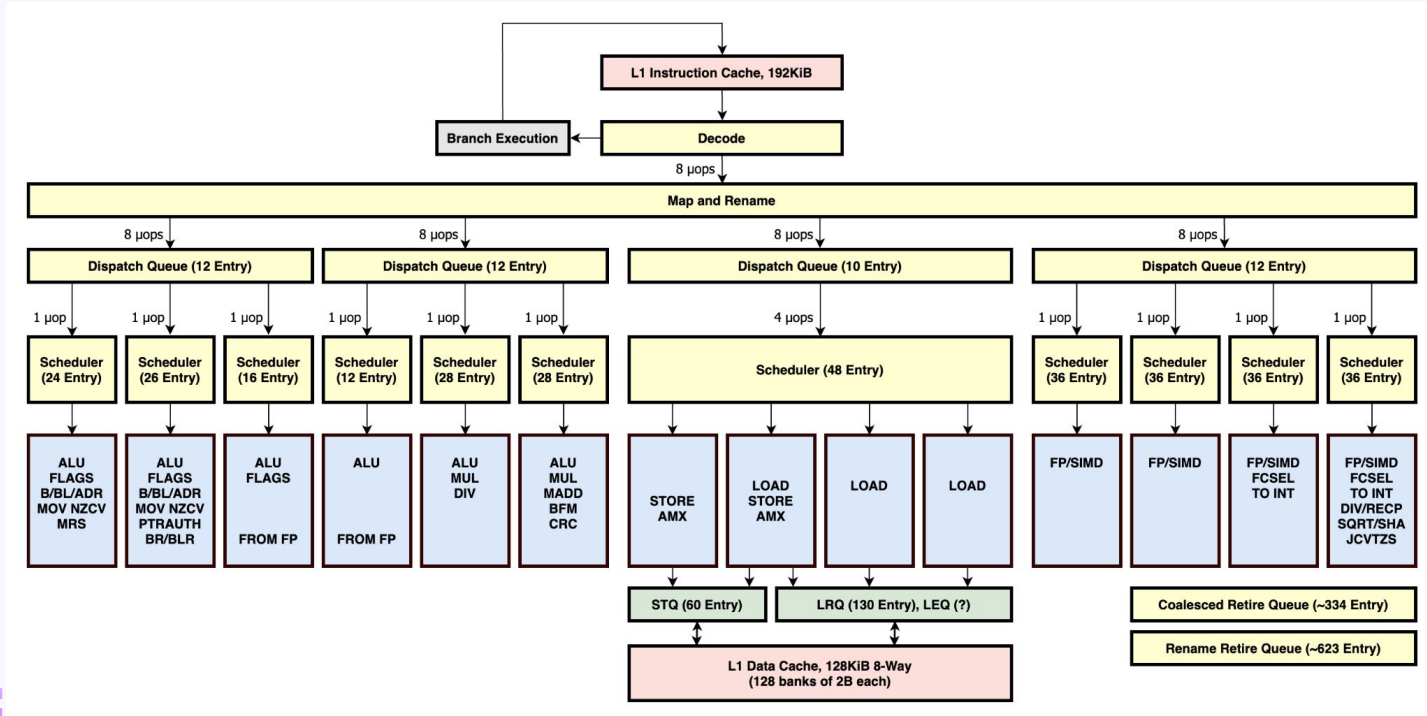
# Intel Core i7 (H&P fig. 3.41)

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



# Firestorm (Apple M1)

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

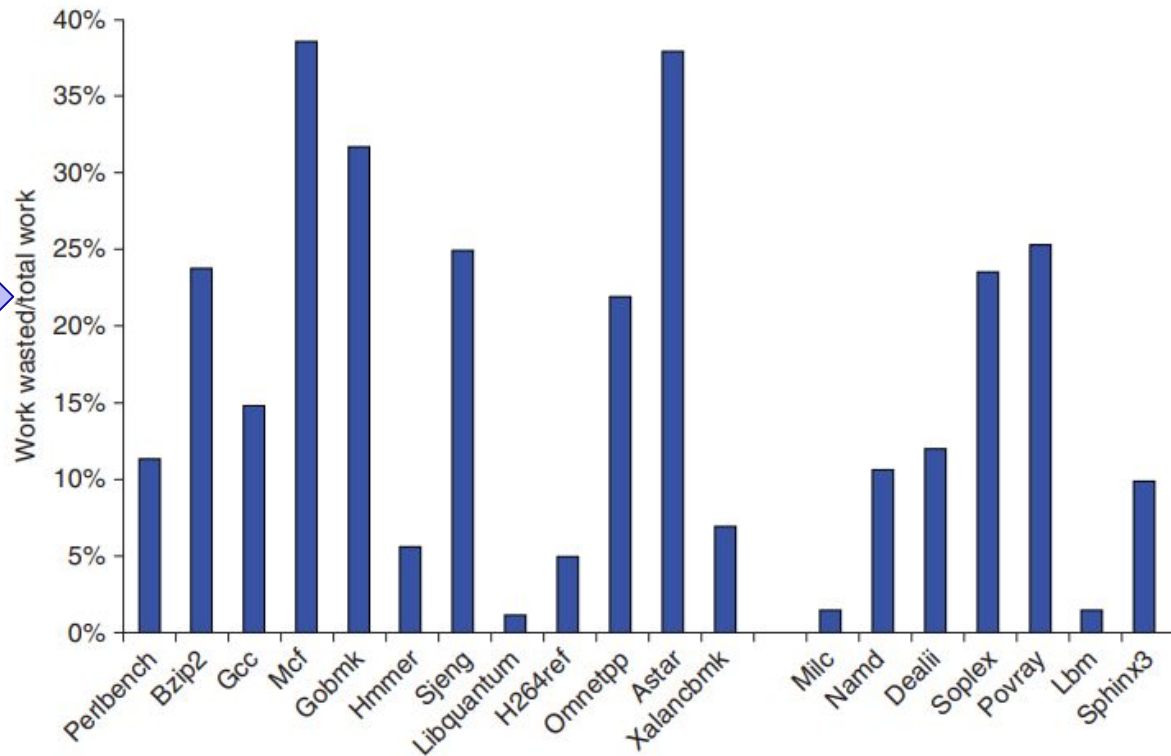




What performance metrics (beyond CPI)  
might become important in a speculative  
CPU?

# H&P fig. 3.42

% of executed  
uops that were  
not committed







What effects would speculative execution have  
on the memory system?  
(Hint: think protected access and/or caches)