

Superscalar (I)

What is superscalar

- A superscalar processor uses multiple independent instruction pipelines. Each pipeline can handle multiple instructions at a time. Multiple pipelines introduces a new level of parallelism, enabling multiple streams of instructions to be processed at a time.
- You can think of a superscalar processor as there are more than one set of washer, dryer, and person who can fold. So, it allows more throughput by processing instructions in parallel.
- Example:
 - Ordinary pipeline: four stages (Fetch, Decode, Execute, Write back), one clock cycle per stage. Executing 6 independent instructions take 9 clock cycles.

I0:	F	D	E	W					
I1:		F	D	E	W				
I2:			F	D	E	W			
I3:				F	D	E	W		
I4:					F	D	E	W	
I5:						F	D	E	W
cc:	1	2	3	4	5	6	7	8	9

- Superscalar: a superscalar that attempts to process 2 instructions simultaneously. Executing 6 independent instructions take 6 clock cycles.

I0:	F	D	E	W		
I1:	F	D	E	W		
I2:		F	D	E	W	
I3:		F	D	E	W	
I4:			F	D	E	W
I5:			F	D	E	W
cc:	1	2	3	4	5	6

- Superscalar supports instruction-level parallelism, while multi-core supports thread or process-level parallelism. Modern processors often use a combination of both superscalar and multi-core techniques to maximize performance. Processors with multi cores may also be superscalar with in each core, allowing for both instruction-level and thread-level parallelism.

Limitations of Superscalar

- A superscalar processor typically fetches multiple instructions at a time and then attempts to find nearby instructions that are independent of one another, and can therefore be executed in parallel like the above superscalar example. However, instructions are usually relevant to each other. Just like the hazards in pipeline, superscalar has limitations too.
- There are five fundamental limitations the system must cope, which are [true data dependency](#), [procedural dependency](#), [resource conflict](#), [output dependency](#), and [anti-dependency](#).

True data dependency (RAW)

- A type of data hazard.
- It happens when the next instruction depends on the result of the previous instruction (aka, the next instruction needs to read the data that is written back by the previous instruction).
- For example, I1 below needs the data stored in R3, which is the data written back to R3 by I0.

I0: ADD R1 R2 R3 # r3 = r1 + r2

I1: MOVE R3 R4 # r4 = r3

- If using a 2-degree superscalar to run these two instructions, the two instructions can be fetched and decoded in parallel. But, the second instruction has to wait till the first instruction finish executing, then start executing by using the forwarding (forwarded data from the first instruction).

I0:F	D	E	W	
I1:F	D		E	W

Procedural dependency

- A type of control hazard.
- It happens when the instructions after a branch can not execute until the branch is executed.
- Can be improved by using branch prediction. If using branch prediction, when to execute the instructions after a branch depends on the prediction.

Resource conflict

- A type of structural hazard.
- It happens when two or more instructions requiring access to the same resource at the same time.
- Can be eliminated by duplication of resources or stalling.

Output dependency (WAW)

- Before talking about the output dependency, let's take a look at the example

I0: R3 op R5 → R3

I1: R3 + 1 → R4

I2: R5 + 1 → R3

I3: R3 op R4 → R7

- Are there any true data dependencies (RAW) we mentioned above in this example?
 - True data dependency: I1 depends on the result of I0, I3 depends on the result of I2, I3 also depends on the result of I1.
- Since I0 and I2 have no true data dependency, how about execute I0 and I2 in parallel, and I1 and I3 in parallel? [No]
- If I2 completes before I0, the contents of R3 will be wrong to I3 and any instructions after I3 that use R3.
- This is what we call “output dependency.” It happens when two instructions write to the same resource.

Anti-dependency (WAR)

- Let's use the same example:

I0: R3 op R5 → R3

I1: R3 + 1 → R4

I2: R5 + 1 → R3

I3: R3 op R4 → R7
- In addition to the true data dependencies and the output dependencies, there is also an anti-dependency.
 - I2 can NOT complete before I0 starts, since I0 needs a value in R3 and I2 changes R3.
 - I2 can NOT complete before I1 starts, since I1 needs a value in R3 and I2 changes R3.
- Anti-dependency happens when an instruction writes to a resource, and the resource must be read by the previous instruction.
- The order of instruction execution is usually assisted by the compiler. The hardware and the compiler assure that parallel execution does not violate the intent of the program.