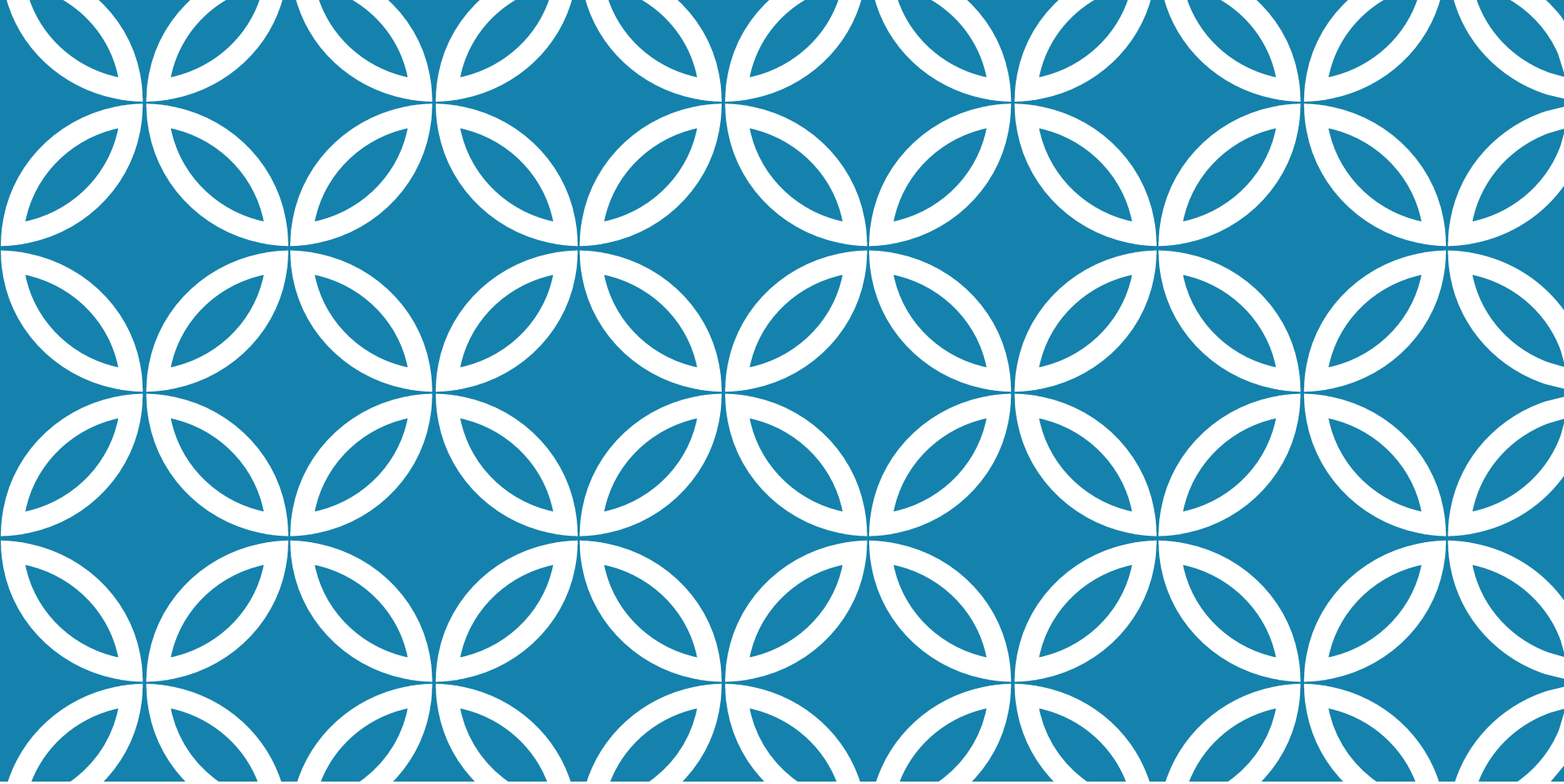
Computer Architecture
Lecture 16



CHAP-16; INSTRUCTION LEVEL PARALLELISM AND SUPER SCALAR PROCESSOR

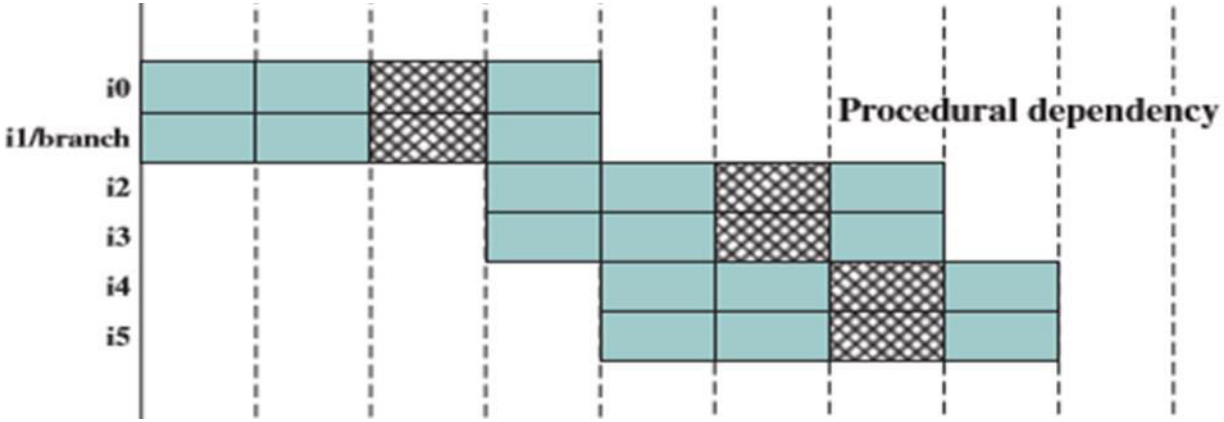
Computer Organization and Architecture Designing for Performance. 9th Edition. William Stallings.

Super Scalar processor Limitations?

- Hazards
  - Procedural dependency (Control Hazards)
  - Resource conflicts (Structural Hazards)
  - Data Hazards
    - True Data dependency (RAW Hazards)
    - Output dependency (WAW Hazards)
    - Antidependency (WAR Hazards)

Procedural Dependency (Control Hazard)

- Can't execute instructions after a branch condition unless decision of branch is made (Taken or NotTaken)?
- Procedural hazards becomes even more severe in Super Scalar processors.
- Performance degrades by the same ratio as it is expected to improve.
- Solution?

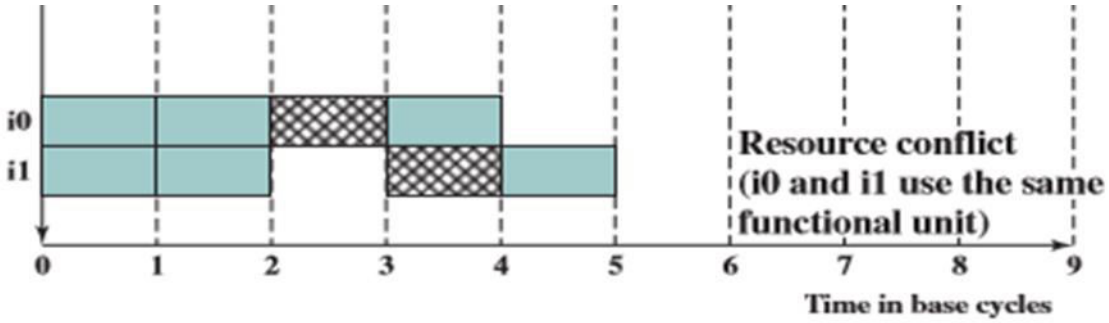


Solution Procedural Dependency (Control Hazard)

- Delay branch not used much as
  - Multiple instructions need to execute in the delay slot.
  - This leads to much complexity.
- Branch prediction should be used
  - Static branch prediction (Power PC 601)
  - Dynamic branch prediction based on Branch history analysis. (Power PC 620)

Resource Conflict (Structural Hazard)

- Two or more instructions requiring access to the same resource at the same time
  - e.g. two arithmetic instructions need the ALU



- Solution - Can possibly duplicate resources
  - e.g. have two arithmetic units, two cache memories, register file ports.

True Data Dependency (Read after Write RAW)

```
ADD   r1, r2     r1+r2 → r1
MOVE  r3, r1     r1 → r3
```

Can fetch and decode second instruction in parallel with first but can't execute. Similarly,

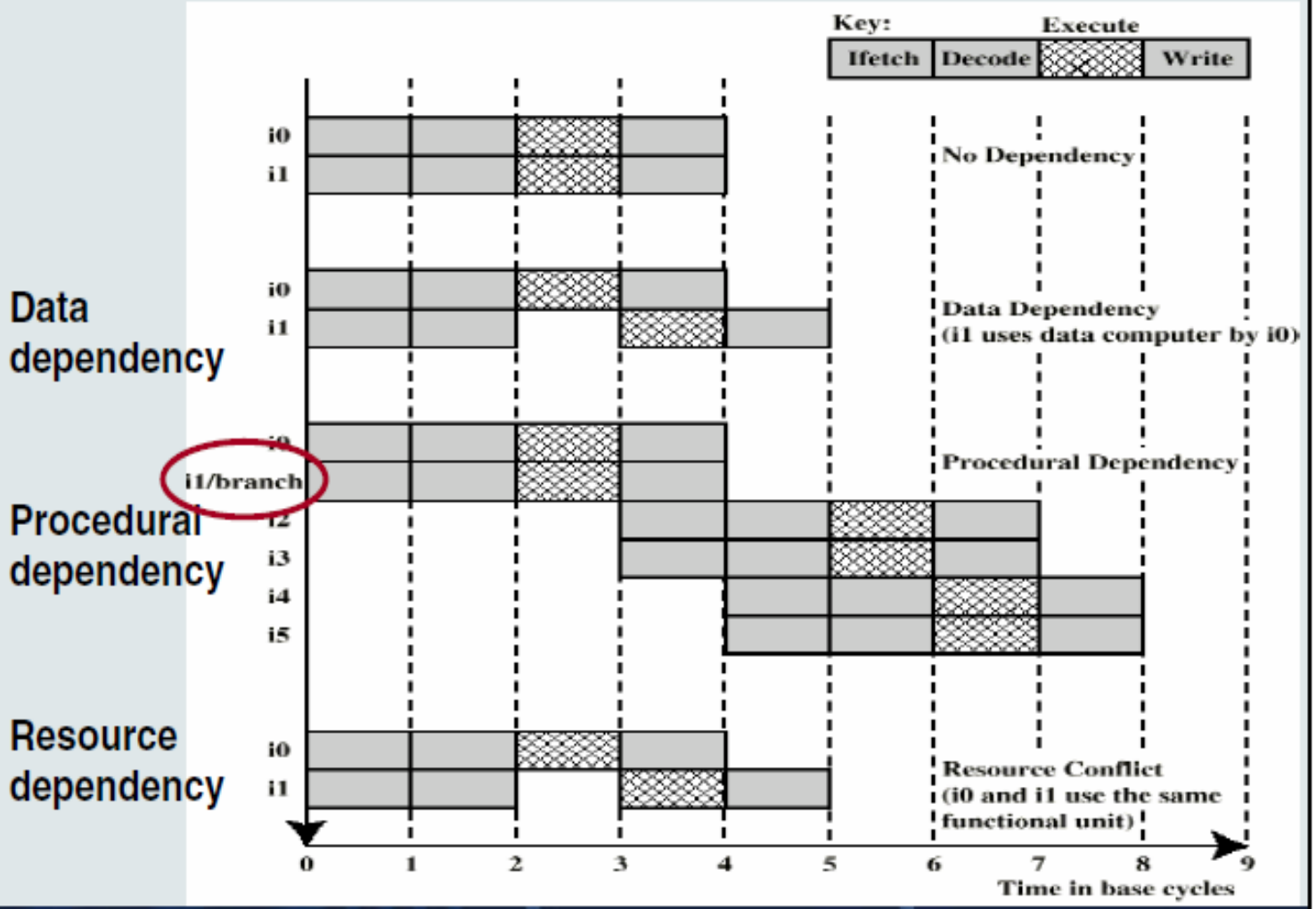
```
LOAD    r1, X     x (memory) → r1
MOVE    r3, r1    r1 → r3
```

Can NOT execute second instruction until first is finished because second instruction is dependent on first (RAW hazard)

Solutions?

- Stall
- Forward
- Code re-order

Effect of Dependencies



Super Scalar processor Design Issues ?

- Design issues
  - Instruction level parallelism
    - There should be sufficient instructions which can be executed in parallel
  - Machine level parallelism
    - Hardware should be capable of executing instructions in parallel
  - Instruction issue policy
    - How instructions should be issued in order to maximize instruction level parallelism?

Instruction vs Machine Parallelism

- **Instruction-level parallelism (ILP)** — the average number of instructions in a program that a processor might be able to execute at the same time.
  - Mostly determined by the number of true (data) dependencies and procedural (control) dependencies in relation to the number of other instructions.
- **Machine parallelism** of a processor — the ability of the processor to take advantage of the ILP of the program.
  - Determined by the number of instructions that can be fetched and executed at the same time, i.e., the capacity of the hardware.
- To achieve high performance, we need both ILP and machine parallelism.
  - The ideal situation is that we have the same ILP and machine parallelism.

Instruction Issue Policies

- Order in which instructions are fetched
- Order in which instructions are executed
- Order in which instructions update registers and memory values (order of completion)

Standard Categories:

- In-order issue with in-order completion
- In-order issue with out-of-order completion
- Out-of order issue with out-of-order completion

In-Order Issue with In-Order Completion

- Instructions are issued in exact program order, and completed in the same order (with parallel issue and completion, of course!).
  - An instruction cannot be issued before the previous one has been issued;
  - An instruction cannot be completed before the previous one has been completed.
- To guarantee in-order completion, an instruction will stall when there is a conflict and when a unit requires more than one cycle to execute.

In-Order Issue -- In-Order Completion

Issue instructions in the order they occur:

- Not very efficient (Not even scalar pipelines use such a simple minded policy)
- Instructions must stall if necessary (and stalling in superscalar pipelining is expensive)

- Instruction issue: When an instruction moves from decode stage to execution stage we say that **instruction is issued.**

In-Order Issue -- In-Order Completion (Example)

- Assume we have 6 instruction to execute (I1, I2, … I6) on a superscalar processor, capable of **fetching/decoding two instructions** per cycle**, executing three instructions** per cycle and can **write two instructions** per cycle. **Processor have two integer units and one floating point unit**
  - I1 requires 2 cycles to execute, rest of the instructions need one cycle to complete.
  - I3 & I4 conflict for the same functional unit
  - I5 depends upon value produced by I4
  - I5 & I6 conflict for a functional unit
- Instructions are fetched in pairs, next two instructions must wait until decode pipeline stage is cleared from previous both instructions.
- To guarantee in order completion, when there is a conflict for functional unit or when functional unit requires more than one cycle to generate result, issue of instructions is stalled.

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8

Takes 8 cycles to complete 6 instructions.

In-Order Issue -- Out-of-Order Completion (Example)

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
	I4	I1		I3	I2		3
I5	I6			I4	I1	I3	4
	I6		I5		I4		5
			I6		I5		6
					I6		7

Again:

- I1 requires 2 cycles to execute
- I3 & I4 conflict for the same functional unit
- I5 depends upon value produced by I4
- I5 & I6 conflict for a functional unit

- Takes 7 cycles to complete six instructions.

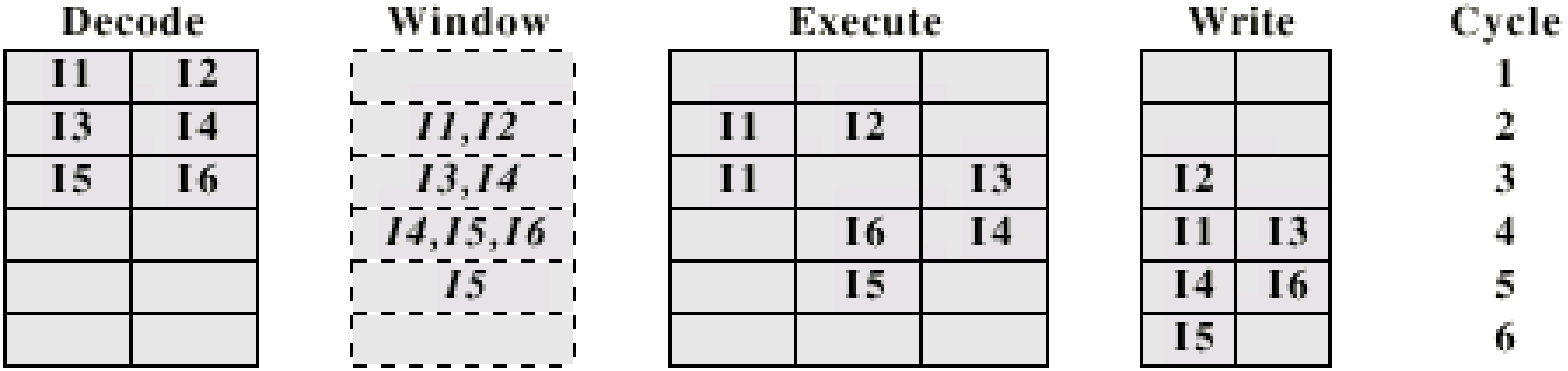
Out-of-Order Issue w. Out-of-Order Completion

- With in-order issue, no new instruction can be issued when the processor has detected a conflict, and is stalled until after the conflict has been resolved.
  - The processor is not allowed to look ahead for further instructions, which could be executed in parallel with the current ones.
- Out-of-order issue takes a set of decoded instructions, issues any instruction, in any order, as long as the program execution is correct.
  - Decouple decode pipeline from execution pipeline, by introducing an instruction window.
  - When a functional unit becomes available an instruction can be executed.
  - Since instructions have been decoded, processor can look ahead.

Out-of-Order Issue -- Out-of-Order Completion

- Decouple decode pipeline from execution pipeline
- Introduce an instruction window (Buffer) to hold decoded instructions temporarily.
- Can continue to fetch and decode until "instruction window" is full.
- When a functional unit becomes available, instruction from instruction window can be passed on to execution stage.
- Since instructions have been decoded, processor can look ahead for new independent instructions.

Out-of-Order Issue -- Out-of-Order Completion (Example)



Decode		Window	Execute			Write		Cycle
I1	I2							1
I3	I4	*I1,I2*	I1	I2				2
I5	I6	*I3,I4*	I1		I3	I2		3
		I4,I5,I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

Again:

- I1 requires 2 cycles to execute
- I3 & I4 conflict for the same functional unit
- I5 depends upon value produced by I4
- I5 & I6 conflict for a functional unit

Note: I5 depends upon I4, but I6 does not so I6 executes before I5 does. Takes six cycles to complete all instructions.

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
I3	I4	I1					3
	I4			I3	I1	I2	4
I5	I6			I4			5
	I6		I5		I3	I4	6
			I6				7
					I5	I6	8

In order issue, in order completion

Decode		Execute			Write		Cycle
I1	I2						1
I3	I4	I1	I2				2
	I4	I1		I3	I2		3
I5	I6			I4	I1	I3	4
	I6		I5		I4		5
			I6		I5		6
					I6		7

In order issue, out of order completion

Decode		Window	Execute			Write		Cycle
I1	I2							1
I3	I4	*I1, I2*	I1	I2				2
I5	I6	*I3, I4*	I1		I3	I2		3
		I4, I5, I6		I6	I4	I1	I3	4
		I5		I5		I4	I6	5
						I5		6

Out of order issue, out of order completion